저작자표시-비영리-변경금지 2.0 대한민국

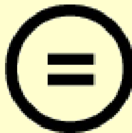
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Ph.D. DISSERTATION

Fast Incremental Density-Based Clustering over Sliding Windows

슬라이딩 윈도우상의 빠른 점진적 밀도 기반 클러스터링

BY

Bogyeong Kim

AUGUST 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Fast Incremental Density-Based Clustering over Sliding Windows

슬라이딩 윈도우상의 빠른 점진적 밀도 기반 클러스터링

BY

Bogyeong Kim

AUGUST 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Fast Incremental Density-Based Clustering over Sliding Windows

슬라이딩 윈도우상의 빠른 점진적 밀도 기반 클러스터링

지도교수 문 봉 기

이 논문을 공학박사 학위논문으로 제출함

2022년 4월

서울대학교 대학원

컴퓨터공학부

김 보 경

김보경의 공학박사 학위 논문을 인준함

2022년 6월

위 원 장: 박 근 수

부위원장: 문 봉 기

위 원: 심 규 석

위 원: 강 유

위 원: 이 종 욱

Abstract

Given the prevalence of mobile and IoT devices, continuous clustering against streaming data has become an essential tool of increasing importance for data analytics. Among many clustering approaches, density-based clustering has garnered much attention due to its unique advantage that it can detect clusters of an arbitrary shape when noise exists. However, when the clusters need to be updated continuously along with an evolving input dataset, a relatively high computational cost is required. Particularly, deleting data points from the clusters causes severe performance degradation.

In this dissertation, the performance limits of the incremental density-based clustering over sliding windows are addressed. Ultimately, two algorithms, $\mathcal{DISC}$ and *DenForest*, are proposed. The first algorithm $\mathcal{DISC}$ is an incremental density-based clustering algorithm that efficiently produces the same clustering results as DBSCAN over sliding windows. It focuses on redundancy issues that occur when updating clusters. When multiple data points are inserted or deleted individually, surrounding data points are explored and retrieved redundantly. $\mathcal{DISC}$ addresses these issues and improves the performance by updating multiple points in a batch. It also presents several optimization techniques. The second algorithm *DenForest* is an incremental density-based clustering algorithm that primarily focuses on the deletion process. Unlike previous methods that manage clusters as a graph, *DenForest* manages clusters as a group of spanning trees, which contributes to very efficient deletion performance. Moreover, it provides a batch-optimized technique to improve the insertion performance. To prove the effectiveness of the two algorithms, extensive evaluations were conducted, and it is demonstrated that $\mathcal{DISC}$ and *DenForest* outperform the state-of-the-art density-based clustering algorithms significantly.

keywords: Density-Based Clustering, Data Streams, Sliding Window, Deletion, Incremental Clustering, DISC, DenForest
student number: 2015-22894

Contents

Abstract i

Contents ii

List of Tables vi

List of Figures vii

1 Introduction 1

1.1 Overview of Dissertation 3

2 Related Works 7

2.1 Clustering 7

2.2 Density-Based Clustering for Static Datasets 8

2.2.1 Extension of DBSCAN 8

2.2.2 Approximation of Density-Based Clustering 9

2.2.3 Parallelization of Density-Based Clustering 10

2.3 Incremental Density-Based Clustering 10

2.3.1 Approximated Density-Based Clustering for Dynamic Datasets 11

2.4 Density-Based Clustering for Data Streams 11

2.4.1 Micro-clusters 12

2.4.2 Density-Based Clustering in Damped Window Model 12

2.4.3 Density-Based Clustering in Sliding Window Model 13

2.5 Non-Density-Based Clustering 14
2.5.1 Partitional Clustering and Hierarchical Clustering 14
2.5.2 Distribution-Based Clustering 15
2.5.3 High-Dimensional Data Clustering 15
2.5.4 Spectral Clustering 16

3 Background **17**
3.1 DBSCAN 17
3.1.1 Reformulation of Density-Based Clustering 19
3.2 Incremental DBSCAN 20
3.3 Sliding Windows 22
3.3.1 Density-Based Clustering over Sliding Windows 23
3.3.2 Slow Deletion Problem 24

4 Avoiding Redundant Searches in Updating Clusters **26**
4.1 The $\mathcal{DISC}$ Algorithm 27
4.1.1 Overview of $\mathcal{DISC}$ 27
4.1.2 COLLECT 29
4.1.3 CLUSTER 30
4.1.3.1 Splitting a Cluster 32
4.1.3.2 Merging Clusters 37
4.1.4 Horizontal Manner vs. Vertical Manner 38
4.2 Checking Reachability 39
4.2.1 Multi-Starter BFS 40
4.2.2 Epoch-Based Probing of R-tree Index 41
4.3 Updating Labels 43

5 Avoiding Graph Traversals in Updating Clusters **45**
5.1 The *DenForest* Algorithm 46
5.1.1 Overview of *DenForest* 47

5.1.1.1 Supported Types of the Sliding Window Model . . 48
5.1.2 Nostalgic Core and Density-based Clusters 49
5.1.2.1 Cluster Membership of Border 51
5.1.3 *DenTree* 51
5.2 Operations of *DenForest* 54
5.2.1 Insertion 54
5.2.1.1 $\mathcal{MST}$ based on Link-Cut Tree 57
5.2.1.2 Time Complexity of `Insert` Operation 58
5.2.2 Deletion 59
5.2.2.1 Time Complexity of `Delete` Operation 61
5.2.3 Insertion/Deletion Examples 64
5.2.4 Cluster Membership 65
5.2.5 Batch-Optimized Update 65
5.3 Clustering Quality of *DenForest* 68
5.3.1 Clustering Quality for Static Data 68
5.3.2 Discussion 70
5.3.3 Replaceability 70
5.3.3.1 Nostalgic Cores and Density 71
5.3.3.2 Nostalgic Cores and Quality 72
5.3.4 1D Example 74

6 Evaluation **76**
6.1 Real-World Datasets 76
6.2 Competing Methods 77
6.2.1 Exact Methods 77
6.2.2 Non-Exact Methods 77
6.3 Experimental Settings 78
6.4 Evaluation of $\mathcal{DISC}$ 78
6.4.1 Parameters 79

6.4.2 Baseline Evaluation 79
6.4.3 Drilled-Down Evaluation 82
6.4.3.1 Effects of Threshold Values 82
6.4.3.2 Insertions vs. Deletions 83
6.4.3.3 Range Searches 84
6.4.3.4 MS-BFS and Epoch-Based Probing 85
6.4.4 Comparison with Summarization/Approximation-Based Methods 86
6.5 Evaluation of *DenForest* 90
6.5.1 Parameters 90
6.5.2 Baseline Evaluation 91
6.5.3 Drilled-Down Evaluation 94
6.5.3.1 Varying Size of Window/Stride 94
6.5.3.2 Effect of Density and Distance Thresholds 95
6.5.3.3 Memory Usage 98
6.5.3.4 Clustering Quality over Sliding Windows 98
6.5.3.5 Clustering Quality under Various Density and Distance Thresholds 101
6.5.3.6 Relaxed Parameter Settings 102
6.5.4 Comparison with Summarization-Based Methods 102

7 Future Work: Extension to Varying/Relative Densities 105

8 Conclusion 107

Abstract (In Korean) 120

List of Tables

4.1 Notations 28

5.1 Time and Space Complexity 46

5.2 Notation 48

5.3 Link-Cut Tree Operations 57

5.4 Clustering quality on various datasets 69

6.1 Threshold values and window sizes 79

6.2 Threshold values and window sizes 91

List of Figures

1.1 Density-Based Clustering over Sliding Windows 2
1.2 Dissertation Overview 4

3.1 Density-based clustering 18
3.2 Graph Representation 19
3.3 Sliding Window Model 23
3.4 Cluster Evolution. Only the *cores* are shown in the figure. The point p denotes a newly added *core*, and the point q denotes a vanishing *core*. 24

4.1 Overview of $\mathcal{DISC}$ 27
4.2 Minimal bonding cores of an *ex-core* p 33
4.3 Cluster evolution by sliding window 34

5.1 Overview of *DenForest* 47
5.2 Example of *DenTree* 53
5.3 DenForest's Insertion 56
5.4 DenForest's Deletion 60
5.5 Insertion/Deletion Example 63
5.6 Example of Super Nostalgic Cores 66
5.7 Batch-optimized Insert 67
5.8 Coverage and Connectivity w.r.t. $|NC_\epsilon|$ 73
5.9 Density-Based Clusters produced by *DenForest* and DBSCAN 74

6.1 Relative speedup over DBSCAN with a varying size of stride 80
6.2 Relative speedup over DBSCAN with a varying size of window . . . 80
6.3 Threshold effects : distance (ϵ) and density (τ) 82
6.4 Insertions vs. Deletions 83
6.5 Range searches executed 84
6.6 Effects of optimizations 85
6.7 Maze: ARI and Update Latency 87
6.8 DTG: ARI and Update Latency 87
6.9 Update Latency with varying ϵ 88
6.10 Illustration of clusters found in Maze ($|W|$=480K) 89
6.11 Illustration of clusters found in DTG ($|W|$=2.56M) 89
6.12 Update Latency (|Stride|/|Window|=5%) 92
6.13 Speedup of `Delete` 93
6.14 Varying size of window (|Stride|/|Window|=5%) 94
6.15 Varying size of stride 95
6.16 Varying ϵ for the DTG dataset 96
6.17 Varying τ for the DTG dataset 96
6.18 Memory usage for various datasets 97
6.19 Memory usage for various window sizes (DTG) 97
6.20 Clustering quality over sliding windows 99
6.21 Clusters found in DTG 99
6.22 Quality under various distance thresholds 100
6.23 Quality under various density thresholds 100
6.24 Relaxed parameter settings 101
6.25 Quality of DTG clusters 103
6.26 Quality of MAZE clusters 103
6.27 Latency with DTG and MAZE 104

7.1 Clusters of Varying Densities 105

7.2 Directions for Future Work 106

Chapter 1

Introduction

Clustering is one of the common methods of unsupervised learning, which discovers the natural groupings in the unlabeled data. Since K-means [61] was proposed more than a half century ago, clustering has been studied extensively (publishing thousands of clustering algorithms in the literature [47]) and applied widely to many data analysis tasks in various fields.

Among many clustering approaches, density-based clustering, pioneered by Ester *et al.* [28] has its unique advantages. Unlike K-means (and another well-known BIRCH algorithm [91]) that discovers spherical clusters, the density-based approach can identify clusters of an arbitrary shape without requiring the pre-set number of labels and can determine which cluster each non-noise data object belongs to. Due to its unique advantages, the density-based clustering garnered much attention. Applications that rely on density-based clustering include the detection of hot spots or segmented regions [30, 89, 72], geo-social network analysis [6, 54], the classification of LiDAR point clouds [21, 31], and mining events by clustering text messages [58].

The density-based clustering is, however, computationally intensive, and the execution of these analytic tasks for time-varying or streaming data involves significant challenges for real-time clustering. Consider a traffic monitoring system that periodically alerts the local public about congested regions. The congested regions (or

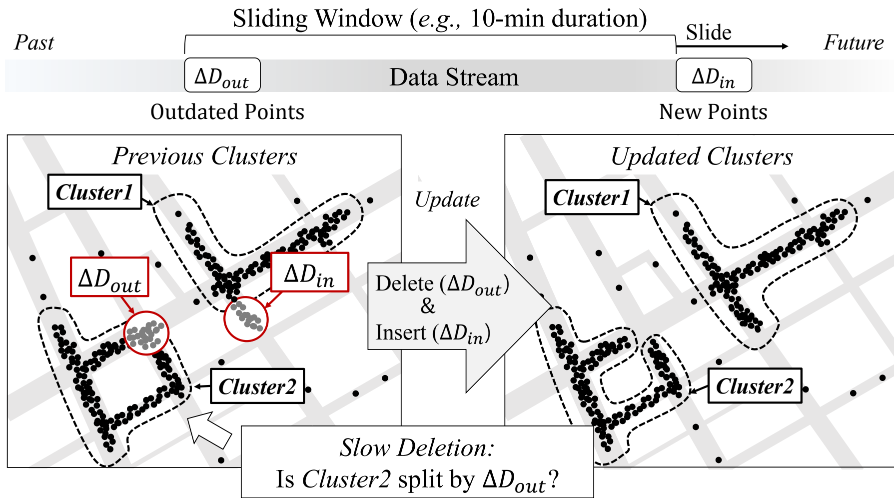


Figure 1.1: Density-Based Clustering over Sliding Windows

density-based clusters) are determined based on the most recent ten-minute vehicular GPS data, which is updated every 30 seconds. The clusters will be reproduced periodically over the sliding window of a 10-min duration that advances every 30s. (See Figure 1.1 for illustration.) To perform this task in a timely manner, the density-based clustering method would update the congested regions incrementally rather than recomputing them from scratch every 30s. Such incremental clustering must update the congested regions efficiently by including new data points (ΔD_{in}) as well as excluding the outdated ones (ΔD_{out}) from the analysis. However, existing incremental density-based clustering approaches require a high computational cost.

The density-based algorithms often manage clusters as a graph either *physically* or *logically*. Representing clusters as a graph is a popular approach, since it enables an algorithm to represent clusters having irregular shapes. However, it requires a costly graph traversal to check whether the cluster is still connected after deleting a point. That is, deleting even a single point can split an existing cluster, and it may require exploring a number of surrounding objects to check the split of the cluster. This task is essentially equivalent to the problem of dynamic graph connectivity [73], and it be-

comes the primary cause of the *slow deletion* by the incremental density-based clustering algorithms [82]. For example, the Incremental DBSCAN algorithm [27] requires numerous spatial range searches to traverse a graph whenever a point is removed. One approach called Extra-N [87] avoids the slow deletion problem by pre-computing clusters in future windows, but it consumes memory too much and suffers from its own slow insert operation when clusters need to be updated frequently.

Grid-based approximation methods have also been reported in the literature [34, 33]. The ρ-double-approximate DBSCAN algorithm [34] achieves poly-logarithmic time complexity for deletion by adopting Holm *et al.* 's data structure [44], which is proposed for dynamic graph connectivity algorithms. However, this grid-based algorithm requires a large number of approximate counting and nearest neighbor queries to such an extent that its practical performance becomes worse than the aforementioned approaches. The performance degradation is even more aggravated when high-resolution clusters are required [71, 84].

Given the relatively high cost of density-based clustering for streaming datasets, many summarization-based approaches have been proposed to expedite the continuous re-discovery of clusters for streaming data [13, 67, 41, 37, 16, 68, 74]. These methods are good at discovering clusters quickly from the infinite data streams; they consume less memory and generally show low latency. However, they cannot capture the clusters accurately in real time and cannot achieve high clustering quality enough to replace the exact approaches such as DBSCAN.

1.1 Overview of Dissertation

The goal of this dissertation is to address the limitation of density-based clustering so that the clustering tasks for streaming data can be carried out in a timely manner without compromising the quality of clustering results or consuming an excessive amount of computational resources. Particularly, the performance degradation in the deletion

[Density-Based Clustering]

Incremental Operations

DISC algorithm (Chapter 4)

COLLECT operation (Section 4.1.2)
- Collect data points to be inserted/deleted

CLUSTER operation (Section 4.1.3)
- Update clusters according to the collected data points
- Efficient update
  - Minimal bonding cores (Def. 9 and Def. 11)
  - Multi-starter BFS (Section 4.2.1)
  - Epoch-based probe (Section 4.2.2)

DenForest algorithm (Chapter 5)

INSERT operation (Section 5.2.1)
- Process data points to be inserted
- Keep DenTree cycle-free
- Batch-optimized update (Section 5.2.5): Exploit locality in updating data points

DELETE operation (Section 5.2.2)
- Process data points to be deleted
- Efficient deletion process (Spanning tree based management) (Section 5.1.3)

Varying/Relative Density

Future Work (Chapter 7)

Figure 1.2: Dissertation Overview

is addressed squarely to accelerate the density-based clustering over sliding windows. Achieving this goal is not an easy challenge to tackle, which has been confirmed repeatedly by the previous studies [27, 82, 34, 87].

Ultimately, two algorithms are presented in this dissertation. (See Figure 1.2.) The first algorithm $\mathcal{DISC}$ is an efficient incremental clustering algorithm over sliding windows that is capable of producing exactly the same clustering results as DBSCAN. $\mathcal{DISC}$ efficiently inserts and deletes data points into/from clusters with `COLLECT` and `CLUSTER` operations. The `COLLECT` operation gathers data points to be inserted or deleted, and the `CLUSTER` operation partially updates the clusters according to the collected points. $\mathcal{DISC}$ primarily focuses on redundancy issues in updating clusters. When multiple points are inserted or deleted individually, surrounding data points are explored redundantly and also retrieved from the spatial index redundantly. $\mathcal{DISC}$ addresses those issues by supporting batch updates equipped with several optimization techniques. $\mathcal{DISC}$ defines *minimal bonding cores* to process multiple data points together. With *Minimal bonding cores*, the full scan of data points is prevented when the clusters are updated, and the redundancy in exploration can be avoided. Moreover, the

insertion process is accelerated by a spatial index-based optimization called *Epoch-Based Probe*, and the deletion process is further accelerated by a heuristic traversal approach called *Multi-Starter BFS*.

The second clustering algorithm *DenForest* is an algorithm that addresses the costly connectivity check problem in density-based clustering. *DenForest* produces similar density-based clustering results to DBSCAN. It supports `Insert` and `Delete` operations to efficiently update clusters. While the previous methods manage clusters as a graph, *DenForest* manages clusters as a group of spanning trees called *DenTrees*. By managing the *DenTrees*, *DenForest* does not require a costly graph traversal to determine the connectedness of clusters. Therefore, it shows very efficient deletion performance. For fast insertion, *DenForest* provides a batch-optimized technique that exploits the locality of the data points.

To prove the practical effectiveness of the two algorithms, extensive experiments were conducted using real datasets. With the extensive experiments, it is demonstrated that the ideas of $\mathcal{DISC}$ and *DenForest* contribute to the performance improvement in various settings and they outperform the state-of-the-art density-based clustering algorithms significantly. In terms of clustering quality, *DenForest* produces a slightly different clustering result from DBSCAN, while $\mathcal{DISC}$ produces the same clustering results as DBSCAN. With theoretical and experimental analysis, however, it is confirmed that the clustering quality of *DenForest* is not overly compromised, and *DenForest* and the DBSCAN algorithm are comparable with respect to the clustering quality.

This dissertation is organized as follows. Chapter 2 presents the related works of the dissertation. Extensive works are covered in this chapter starting from traditional density-based clustering to non-density-based clustering. Chapter 3 presents the background of the traditional density-based clustering algorithms. It explains the DBSCAN algorithm and the Incremental DBSCAN algorithm in detail. Also, this chapter introduces the task of density-based clustering over sliding windows along with its chal-

lenge. Chapter 4 explains the first algorithm $\mathcal{DISC}$ and Chapter 5 explains the second algorithm *DenForest*. Chapter 6 provides the performance evaluations of the two algorithms. Lastly, Chapter 7 and 8 present future work and conclusion of the dissertation, respectively.

Chapter 2

Related Works

In this chapter, various works related to this dissertation are covered. First, clustering (or cluster analysis) is introduced briefly in Section 2.1. Since clustering has been studied extensively for decades, numerous clustering algorithms have been developed. Among them, density-based clustering, one closely related to the dissertation, is explained in Sections 2.2-2.4. Those sections are organized according to the characteristics of datasets that algorithms assume. Finally, clustering methods besides density-based clustering are briefly introduced with the several major algorithms in Section 2.5.

2.1 Clustering

As the volume and variety of data increased, it became infeasible to manually analyze all those data due to the limited human resources and time. Therefore, there has been a long history of automatically understanding and processing data, and numerous methods have been studied and developed.

Clustering, one of the methods in the data analysis, is the task of grouping data points into several groups so that the data points in the same group get to be more similar to each other than points in other groups [42]. Since clustering finds patterns or groups using the information in the data itself and does not require external in-

formation, it is classified as unsupervised learning in pattern recognition. (Supervised learning requires labels for each point to be classified.) For a long time, clustering has been used in research and industrial fields with various purposes such as gaining insight or detecting anomalies.

2.2 Density-Based Clustering for Static Datasets

One popular group of clustering algorithms is density-based clustering. Density-based clustering algorithms define a cluster as a group of close data points in a dense region. In this section, density-based clustering algorithms for static datasets are covered.

DBSCAN [28] is one of the popular density-based clustering algorithms. DBSCAN takes two parameters, density threshold (τ) and distance threshold (ϵ), to define the density of the region. Then, DBSCAN constructs each cluster by connecting data points in dense regions. (See Section 3.1 for detailed explanation.) DBSCAN has unique advantages that it can detect clusters of irregular shapes when there exists noise in the dataset. Due to its advantages, it has been widely used in various applications [30, 89, 72, 54, 21, 31, 58].

The Jarvis-Patrick algorithm [49] is an algorithm similar to DBSCAN in that it considers neighboring points to construct the clusters. However, while DBSCAN uses neighboring points to define the density of the space, Jarvis-Patrick uses neighboring points to define the similarity for each pair of data points.

2.2.1 Extension of DBSCAN

Since DBSCAN uses the single density threshold value to define the dense region, it is hard to detect meaningful clusters when clusters have different densities. The OPTICS [5] algorithm was proposed to address this weakness. OPTICS produces a reachability plot containing information about cluster structure. In the plot, points are ordered based on their density and proximity. With this plot, users can select appro-

priate clusters having different densities. Note that OPTICS does not directly produce clustering results, but it provides structural information of the data points (*i.e.*, reachability plot) and requires users' intervention. Another work called HDBSCAN [12] was proposed to address this problem. While OPTICS leaves users to select appropriate clusters, HDBSCAN automatically detects the clusters without users' intervention by introducing a concept called cluster stability.

Unlike the above density-based clustering algorithms that assume spatial data points, Sander *et al.* proposed GDBSCAN [70] which generalizes DBSCAN to the dataset having spatial and non-spatial attributes. By generalizing density and reachability concepts to consider the non-spatial attributes, it is applied to various application domains such as astronomy or molecular biology.

It is also worth noting that several works based on sampling are proposed [59, 48, 51]. These works adopt sampling strategies to reduce the computational cost of DBSCAN in large datasets.

2.2.2 Approximation of Density-Based Clustering

Recently, Gan *et al.* proposed an approximated DBSCAN called ρ-approximate DBSCAN [33]. By introducing an approximation factor to the distance threshold (ϵ), ρ-approximate DBSCAN can produce approximate density-based clusters in amortized O (N) time. ρ-approximate DBSCAN manages data points into grids and exploits approximate range counting queries [7] for efficiency. Although Schubert *et al.* [71] confirmed that its practical performance was not as good as expected, ρ-approximate DBSCAN affected various works of parallelization of DBSCAN. Gan *et al.* also proposed approximate OPTICS [35] by formalizing a valley concept in OPTICS and introducing an approximation factor into the concept.

2.2.3 Parallelization of Density-Based Clustering

One of the earliest works for parallization of DBSCAN is PDBSCAN [86] which performs DBSCAN over a distributed system based on master-slave architecture. It proposes a grid partitioning technique for DBSCAN to be performed in parallel. By adopting the grid partitioning technique, MR-DBSCAN [43] is also proposed. Different from PDBSCAN, its algorithm is based on a map-reduce framework, Hadoop, to reduce the bottleneck in the master node of the master-slave architecture. NG-DBSCAN is one of the early works developed on the Spark framework as a scalable solution to density-based clustering [60]. RP-DBSCAN is another parallel DBSCAN algorithm based on the Spark framework that takes advantage of the random split strategy [77]. Wang *et al.* have proposed several exact and approximate DBSCAN algorithms based on grid construction and solving the bichromatic closest pairs problem in parallel [84]. There also exists parallel work for OPTICS and HDBSCAN [85].

2.3 Incremental Density-Based Clustering

Since the DBSCAN algorithm was proposed more than two decades ago, much research has been conducted to make the density-based clustering a viable option even for time-evolving or streaming data. The first one to note is Incremental DBSCAN [27] which updates existing clusters upon each individual data point being inserted into or deleted from the database. By examining and updating only surrounding data points of the inserted or deleted points, it does not re-process the whole data points from scratch. However, it suffers serious performance degradation when a point is deleted from its cluster.

The EXTRA-N algorithm is the one proposed to efficiently detect density-based clusters over sliding windows. In addition to distance and density thresholds (which are the parameters of DBSCAN), EXTRA-N takes another two parameters related to sliding windows, window size $|W|$ and stride size $|S|$. To avoid processing a large

number of range searches required for dealing with deleted data points, it maintains $\frac{|W|}{|S|}$ number of future windows. Although Extra-N [87] avoids the slow deletion problem by pre-computing clusters in future windows, it consumes memory too much and suffers from its own slow insert operation when clusters need to be updated frequently. This will lead to serious performance degradation when the clusters within the sliding window are updated frequently with a relatively small stride.

2.3.1 Approximated Density-Based Clustering for Dynamic Datasets

A dynamic version of ρ-approximate DBSCAN (called ρ-double-approximate DBSCAN) has also been reported in the literature [34]. Similar to the static version [33], it introduces an approximation factor to the distance threshold (ϵ). However, while the static version uses the approximation only to connect core points, the dynamic version uses additional approximation to classify points. By approximately counting the number of neighbors or approximately finding the nearest point, it produces an approximate result of DBSCAN. The ρ-double-approximate DBSCAN algorithm [34] achieves poly-logarithmic time complexity for insertion and deletion by adopting Holm *et al.* 's data structure [44] for the dynamic graph connectivity algorithms. However, due to a large constant number of approximate counting and nearest neighbor queries, its practical performance becomes worse than the previous approaches.

2.4 Density-Based Clustering for Data Streams

Many density-based clustering algorithms have been proposed to expedite the clustering for streaming data. Generally, these algorithms assume an infinite length of data streams but a finite memory capacity.

2.4.1 Micro-clusters

One important concept in clustering for data streams is *micro-cluster* which was proposed in CluStream [1]. A micro-cluster is a temporal extension of the *cluster feature* vector in BIRCH [91]. Due to an infinite length of data streams, clustering algorithms cannot retain all the data points in memory. Instead, they manage statistical information (*i.e.*, micro-cluster) representing a group of close data points. Micro-clusters decays over time to capture the recent information in the data streams. Therefore, recent micro-clusters have more weight than old micro-clusters. By managing the micro-clusters that are far fewer than the whole points, stream clustering algorithms can produce clusters quickly with less memory.

With the micro-clusters, CluStream runs in following two steps.

1. **(Online) Summarization step**. It generates and manages micro-clusters from streaming data with decaying density.
2. **(Offline) Clustering step**. A modification of the k-means algorithm is applied to the micro-clusters to construct the final clustering results.

CluStream has affected numerous works such that most clustering algorithms for data streams run in the above two steps.

2.4.2 Density-Based Clustering in Damped Window Model

One fundamental work in density-based clustering for data streams is DenStream [13]. Similar to CluStream, it adopts the micro-cluster technique and assumes the damped window model where the weight (importance) of the data points decays over time. Different from CluStream, it applies a modification of DBSCAN algorithm in the clustering step. Therefore, DenStream can detect clusters of irregular shapes.

DBSTREAM [41] is one popular work affected by DenStream. Different from DenStream, it introduces a concept called shared density to improve the quality. In

the clustering step, DenStream assigns the same cluster membership to two micro-clusters if they are close enough. However, since each micro-cluster is the statistical information of a group of close points, the micro-clusters' proximity does not ensure the proximity of the points in the micro-clusters. Therefore, DBSTREAM manages additional information about shared density between micro-clusters. If the shared density between two micro-clusters is high enough, DBSTREAM assigns the same cluster membership to them. With this concepts, it achieves higher quality [14].

EDMStream [37] is another method that uses micro-clusters. Different from the above algorithms, it does not adopt the modification of DBSCAN to cluster the micro-clusters. Instead, it adopts the density-peak algorithm and manages the clustering result incrementally for efficiency. The density-peak algorithm [69] is an algorithm based on the idea that cluster centers are surrounded by neighboring points in lower density and that the centers are relatively distant from any points in a higher density.

Instead of using the micro-cluster concept, some works exploit grids. D-Stream [17], one of them, manages grids instead of the micro-clusters. Data points are mapped into the grids of which weight decays over time. To keep the distributional information of the data points in each grid, D-Stream manages an information vector called attraction for each grid. Then, D-Stream uses the attraction vectors in the clustering step.

2.4.3 Density-Based Clustering in Sliding Window Model

While the above works assume the damped window model, some works are assuming the sliding window model. The Exponential Histogram of Cluster Feature (EHCF) technique is widely used to construct a set of micro-clusters in the sliding window model, which was proposed in SWClustering [93]. In the decaying model, very old points and recent points can be grouped into one micro-cluster if they are close enough. However, in the sliding window model, those close data points should not be managed in the single micro-cluster since one micro-cluster cannot be split due to the lack of information about data points to be deleted. To address this issue, EHCF provides

temporal boundaries among micro-clusters that prevents old and new data points from being grouped together. SDStream [68] is one work that adopts the EHCF technique for density-based clustering over sliding windows. A recent work, StreamSW [74], also adopts the EHCF technique, but it is based on the density-grids proposed in D-Stream rather than micro-clusters.

The above algorithms for data streams (which will be called summarization-based methods in this dissertation) are good at discovering clusters quickly from the infinite data streams; they consume less memory and generally show low latency. However, since they only maintain coarse-grained information (*i.e.*, micro-clusters), they cannot capture the clusters accurately in real time and cannot achieve high clustering quality enough to replace the exact approaches such as DBSCAN.

2.5 Non-Density-Based Clustering

In this section, several clustering algorithms besides the density-based clustering algorithms are introduced briefly.

2.5.1 Partitional Clustering and Hierarchical Clustering

Two distinct clustering categories are partitional clustering and hierarchical clustering [47]. Partitional clustering algorithms [61, 66, 53] partition data points into separate groups without hierarchy. (All groups have the same level.) K-means [61] is one of the famous example of partitional clustering algorithms. K-means partitions data points into a fixed number of groups such that within-cluster variances are minimized.

Hierarchical clustering algorithms [75, 23, 38, 12, 91] recursively partition data points into nested groups, and a set of groups can be represented with a dendrogram. Single-linkage [75] and complete-linkage [23] clustering algorithms are popular examples. They construct the nested structure by grouping the closest pair of groups (or clusters) in an agglomerative way. The difference between them is the way they define

the distance between clusters. While the single-linkage algorithm uses the closest distance between two elements (one in each cluster) as the distance between clusters, the complete-linkage algorithm uses the farthest distance between two elements. While single-linkage or complete-linkage algorithms use all points in the clusters to compute the distance between two clusters, BIRCH [91] or CURE [38] algorithms use representative vectors to compute the distance between two clusters. BIRCH uses one representative vector called cluster feature vector for each cluster, while CURE uses multiple representative vectors well scattered for each cluster.

2.5.2 Distribution-Based Clustering

Distribution-based clustering algorithms are grouping points based on the assumption that the data points are generated from a distributional model. Gaussian Mixture Model (GMM) [29, 88] is one of the popular algorithms. GMM assumes that data points are generated from the mixture of gaussian distributions. With the Expectation-Maximization (EM) method [64], parameters of the gaussian distributions are adapted to the data points.

2.5.3 High-Dimensional Data Clustering

In high-dimensional spaces, there exist various problems that do not occur in low-dimensional spaces, which is called the curse of dimensionality. For example, the distance concept gets less precise as the dimensionality increases. Specifically, the distance to the closest data point becomes similar to the distance to the farthest data point. In addition, the high-dimensional space is difficult to visualize, and since there are many possible subspaces, the interpretability of the clustering result deteriorates.

In the clustering fields, there are some works to overcome those problems [3, 11, 2]. One popular algorithm is CLIQUE [3]. The CLIQUE algorithm automatically finds the subspaces that allow better clustering. It is based on the idea that if points are clustered in the k-dimensional space, they are also clustered in the (k-1)-dimensional

projections of this space.

2.5.4 Spectral Clustering

Spectral clustering algorithms [40, 4, 65] detect clusters based on spectral graph theory. Spectral clustering algorithms represent data points as a weighted graph (or an adjacency matrix). Vertices are set to the data points and the weight of each edge is set to the similarity between two vertices. Then, it finds clusters by partitioning the graph with the cut size. This is done by computing the eigenvectors of the $Laplacian$ matrix. Based on the idea that eigenvectors have the structural information of the graph, they find densely connected points (*i.e.*, cluster) by examining each value in the eigenvectors.

Various clustering algorithms are introduced in this chapter. However, note that there is no one best clustering algorithm. Meaningful clusters found by one clustering algorithm can become meaningless as their applications change. Some applications may prefer to pass a parameter indicating the number of clusters. Another applications may prefer to get the clusters having non-spherical shapes. Therefore, an appropriate clustering algorithm is not determined solely by how it defines clusters but depends on applications.

Among various clustering methods, density-based clustering is the one related to this dissertation. Particularly, incremental algorithms and streaming algorithms are the most related works, and they will be the competitors of the proposed algorithms in this dissertation.

Chapter 3

Background

This chapter provides readers with the background information of density-based clustering method and the key characteristics of sliding window models commonly adopted for streaming data processing. The algorithms proposed in this dissertation leverage the sliding window model innovatively and overcome the critical weaknesses of the existing density-based clustering algorithms.

3.1 DBSCAN

The density-based clustering was pioneered by Ester *et al.* more than two decades ago [28]. In this seminal work, the density of a point is defined by the number of neighbors that are within a given distance threshold denoted by a parameter ϵ. It is the density that determines the status of a point as one of *core*, *border*, and *noise*. (See Figure 3.1 for illustration.) Such classification of points is done by introducing another parameter called *MinPts*. If a point has its density no less than *MinPts*, it is labeled as *core*. If a point has its density less than *MinPts* but is within the distance threshold ϵ from at least one other *core* point, it is labeled as *border*. Otherwise, the point is labeled as *noise*. For example, in Figure 3.1, points X, Y, Z are a core point, a border point and a noise, respectively, assuming the density threshold is four.

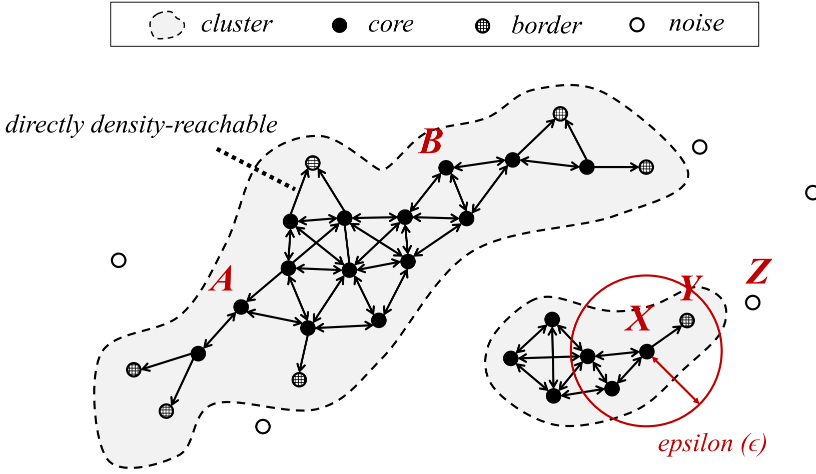


Figure 3.1: Density-based clustering

Ester *et al.* 's DBSCAN algorithm defines a cluster as a set of core and border points that are *density-reachable* from an arbitrary core point of the cluster. Let $N_\epsilon(p)$ denote a set of points within the threshold distance ϵ from p. A point q is said to be *directly density-reachable* from p if $q \in N_\epsilon(p)$ and p is a core.

Definition 1. *A point p is directly density-reachable from a point q if q is a core point and $p \in N_\epsilon(q)$.*

Note that q does not have to be a core point. The direct density-reachability is a symmetric relation for core points, although it is not when a border point is involved. In general, a point q is said to be *density-reachable* from p if there is a chain of directly density-reachable cores from p to q.

Definition 2. *A point p is density-reachable from a point q if there is a chain of points $p_1, p_2...., p_n, p_1 = p, p_n = q$ such that p_i is directly reachable from p_{i+1}.*

In Figure 3.1, border Y is directly density-reachable from core X, but not vice versa. Cores A and B are density-reachable from each other, whereas cores A and X are not. The density-reachability is a transitive but asymmetric relation.

Based on the density-reachability relation of cores and borders, the DBSCAN algorithm attempts to find density-based clusters. Specifically, for each core point p found in the seeding phase, a singleton cluster, say $\mathcal{C}$, containing p is created. Then, in the growing phase, all the directly density-reachable points from any $q \in \mathcal{C}$ are added to $\mathcal{C}$. This process is repeated until $\mathcal{C}$ does not grow any more. Therefore, when it terminates, the DBSCAN algorithm returns density-based clusters, each of which is a *maximally connected component* of core points and border points.

Definition 3. *A $cluster$ C is a non-empty subset of points in dataset, D, satisfying two conditions:*

1. *Maximality: $\forall p, q \in D$: if $p \in C$ and q is density-reachable from p, then $q \in C$.*
2. *Connectivity: $\forall p, q \in C$: there exists a point o from which p and q are reachable.*

3.1.1 Reformulation of Density-Based Clustering

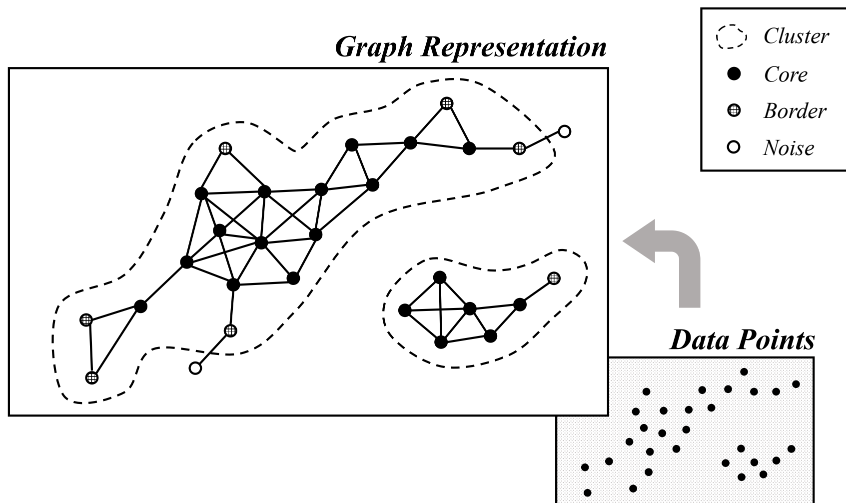


Figure 3.2: Graph Representation

Given the way clusters are defined by DBSCAN, the density-based clustering ($\mathcal{DC}$) can be reformulated as the problem of finding the connected components in a graph [50, 60]. In the graph representation, each vertex corresponds to a data point and an edge is added to a pair of data points if they are within the ϵ-distance from each other. (See Figure 3.2.) Each vertex is then labeled with one of *core*, *border*, or *noise* as described above. Finally, a connected component of *cores* as well as the *borders* adjacent to the connected component is identified as a cluster.

Note that the edges in the graph are not physically stored and managed, since it would require massive memory space at most $O(|V|^2)$. Instead, range queries of a spatial index are used for each vertex to retrieve the adjacent vertices, which plays the role of the edges.

3.2 Incremental DBSCAN

The DBSCAN algorithm was invented for a static database. If an input database is dynamic, the clustering result has to be adjusted whenever a new data point is inserted to or an existing data point is deleted from the database. Ester *et al.* proposed an *incremental* version of the DBSCAN algorithm based on the observation that changes in the clustering result are limited to the neighborhood of an inserted point or a deleted point [27]. Hereinafter it is referred to as *IncDBSCAN* in short.

In general, on the insertion or deletion of p, the points in $N_\epsilon(p)$ or that are density-reachable from those in $N_\epsilon(p)$ may be affected by the update. For a point $q \in N_\epsilon(p)$, the core status of q can be altered by the insertion or deletion of p. The altered core status of q can in turn affect the density reachability between the core points outside $N_\epsilon(p)$. Therefore, IncDBSCAN can limit the scope of examination to the core points in $N_\epsilon(x)$ for any point x that gains or loses its core status as a result of the update (that is, the insertion or deletion of p). This scope of examination is referred to as the *seed objects for insertion* or the *seed objects for deletion* depending on the type of an

update. Formally, the set of seed objects are defined as follows.

- The set $UpdSeed_{Ins}(p)$ of seed objects for p being inserted is $\{q \in N_\epsilon(x) \mid q \text{ is a core } \wedge\ x \text{ gains its core status by } p\}$.
- The set $UpdSeed_{Del}(p)$ of seed objects for p being deleted is $\{q \in N_\epsilon(x) \mid q \text{ is a core } \wedge\ x \text{ loses its core status by } p\}$.

The insertion of p may cause two or more clusters to merge due to the new density-connections created by p. To be precise, new density-connections are created when a point gains its core status by p.

Thus, a *range search* has to be performed for each of the points that gain the core status to find out whether different clusters are now connected by the new core point. A cluster merger may happen when $UpdSeed_{Ins}(p)$ contains core points that belong to two or more clusters.

Similarly, the deletion of p may cause a cluster to split into two or more clusters due to the density-connections destroyed by p. A cluster split happens when $UpdSeed_{Del}(p)$ contains two or more connected components. Thus, a *breadth-first search* has to be performed against all the core points in $UpdSeed_{Del}(p)$ to determine whether they still constitute a single cluster.

For the insertion and deletion of p alike, the scope of examination is limited to the core points in $UpdSeed_{Ins}(p)$ and $UpdSeed_{Del}(p)$, respectively. However, it is critically important to note that the number of range searches required by a deletion is likely to be much higher than that required by an insertion. This is because the breadth-first search performed against $UpdSeed_{Del}(p)$ needs to access all the points in the set and the cardinality of the set is usually much larger than the number of points that would gain their core status by an insertion.

3.3 Sliding Windows

The fundamental premise of computations over data streams is that the streaming data cannot be stored and processed in its entirety by virtue of their sheer volume. One of the popular models for streaming data analytics is the *sliding window* model, which is typically characterized by two parameters known as $window$ and $stride$ [8, 32, 78, 22, 94]. The size of the $window$ defines the range of streaming data to be analyzed, and the $stride$ defines the interval at which the result of the analysis is updated.

Definition 4 (**Window**). *The window W is a set of the latest data points. The size of the window, $|W|$, can be bounded by either the number of points in the window (count-based window) or the duration of the window (time-based window).*

Definition 5 (**Stride**). *The stride S is defined by an interval at which the clustering result is updated while the window slides. The size of the stride, $|S|$, is bounded by the number of points or by a time duration depending on the type of the sliding window. The data points in the same stride are processed together.*

In this model, the one end of the window is assumed to be anchored to the current time or the current data item. This model thus allows us to analyze and understand the most recent data within the current window. Whenever the sliding window advances, some of the existing older data objects (in the oldest stride) will leave the window while newer data objects (in the new stride) will enter it. (See Figure 3.3.)

From the computational point of view, it is important to understand that a multitude of data objects enter or leave the window at once when the window advances and there is no particular order of processing among the data points in the same stride. Furthermore, unlike a *decaying data* model [19] (or damped window model) where the influence of each data object wanes over time, all the data objects in the current window are assumed to carry the same influence or weight.

The sliding window model can be either *time-based* or *count-based* depending on how the two parameters, $window$ and $stride$, are interpreted. While the parameters

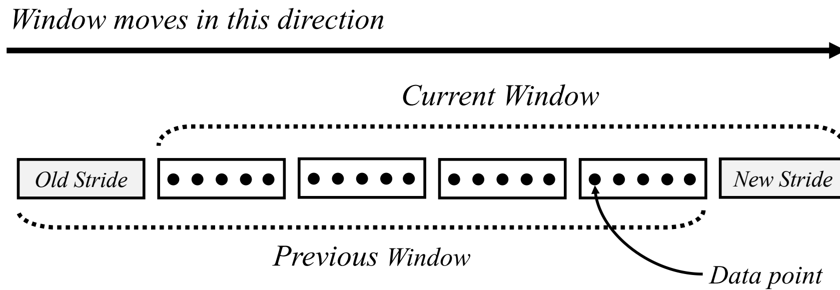


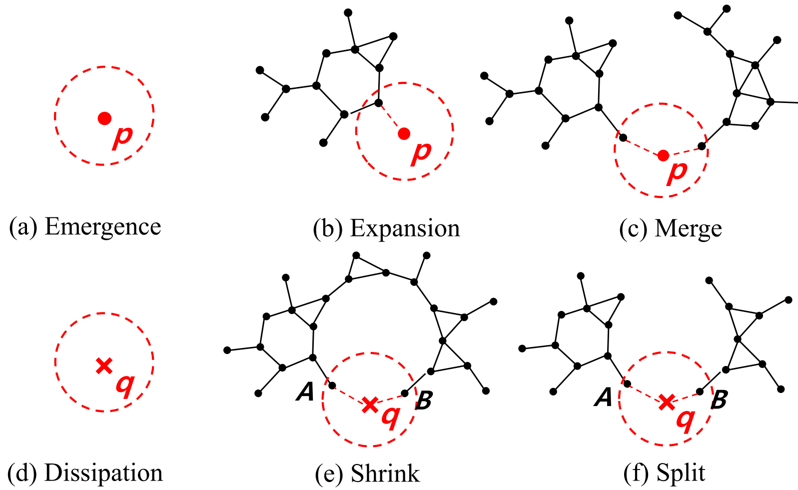
Figure 3.3: Sliding Window Model

are measured in time duration under the former model, they are measured in the number of data objects under the latter. Suppose for example that the sliding window is time-based and its stride is set to 30 seconds. (The window size can be any time duration longer than 30 seconds.) Then, whenever the window slides by 30s, a group of new points added to the window during the period are processed together. Similarly, a group of old points removed from the window during the same period are processed together. The clustering methods proposed in this dissertation is not subject to how those parameters are measured and will work with either of the two model types.

3.3.1 Density-Based Clustering over Sliding Windows

The goal of this work is to find density-based clusters in streaming data under the sliding window model. The sliding window model is widely used to capture the recent state of streaming data, which cannot be stored in its entirety by virtue of the large and ever-increasing volume [32, 22]. As is stated in Section 3.1.1, this task is equivalent to the problem of finding the connected components of cores and their adjacent borders *continuously* as the window advances.

There are six types of cluster evolution that can occur with a sliding window: *emergence, expansion, merge, dissipation, shrink* and *split*, as depicted in Figure 3.4. As new points are inserted into the window, some of the existing points may become *cores*. Due to those new cores, a new connected component ($\mathcal{CC}$ in short) may emerge



(a) Emergence (b) Expansion (c) Merge

(d) Dissipation (e) Shrink (f) Split

Figure 3.4: Cluster Evolution. Only the *cores* are shown in the figure. The point p denotes a newly added *core*, and the point q denotes a vanishing *core*.

or an existing $\mathcal{CC}$ may expand. If any of those new cores connects separate $\mathcal{CC}$s, then they are merged into one. At the same time, old points may be deleted from the window, and some of the existing cores may become *non-cores*. Due to those vanishing cores, existing $\mathcal{CC}$s may shrink or dissipate. A vanishing core may also split a $\mathcal{CC}$ into multiple $\mathcal{CC}$s.

3.3.2 Slow Deletion Problem

The incremental management of clusters involves processing expiring (or vanishing) cores. This becomes a major bottleneck in updating density-based clusters incrementally and increases the latency significantly.

Problem 1 (Slow deletion). *A cluster may be split by a vanishing core. To determine whether the cluster is split or not, the remaining cores need to be traversed to check the density-reachability from one another. This traversal would require a number of range searches, which is the main cause of the degraded performance of incremental density-based clustering.*

Example 1. *Consider a core point q that expires to become a $non\text{-}core$ (either border or noise) point due to some other data points leaving the window. The vanishing core q will trigger one of the three types of evolution, namely dissipation, shrink, and split, as is shown in Figures 3.4(d)-(f). Let $\mathcal{CC}$ denote the connected component of q's cluster. Then a graph traversal such as Breadth-First Search (BFS) can be applied to check the connectedness of $\mathcal{CC} \setminus \{q\}$. In Figures 3.4(e) and 3.4(f), the density-reachability between the two adjacent cores A and B need be checked when q vanishes. This will require visiting all the points in the figure inevitably. In general, if a graph traversal is initiated from one of the cores in $N_\epsilon(q)$ and it can visit all the cores in $N_\epsilon(q)$, then the cluster shrinks but does not split.*

This is equivalent to the dynamic connectivity problem, which has been studied for decades [73, 44]. Due to the $O(N^2)$ memory cost, the edges between core points are often maintained only logically. Thus, density-based clustering methods generally rely on a spatial index to discover a pair of adjacent cores. Hence, the latency concern is further aggravated due to the increasing cost of range searches, when the dimensionality of data points increases.

Chapter 4

Avoiding Redundant Searches in Updating Clusters

The density-based clustering method presented in this chapter is called *Density-based Incremental Striding Cluster* ($\mathcal{DISC}$ in short). It is capable of producing exactly the same clustering results as existing algorithms such as Incremental DBSCAN much more quickly and efficiently. The elaborate design of $\mathcal{DISC}$, based on the novel ideas for the *minimal bonding cores* of *ex-cores* and *neo-cores*, enables it to avoid a considerable amount of redundant work by checking the density-connectedness only for the minimal bonding cores. The *MS-BFS* strategy and the epoch-based R-tree index probing method are proposed to further reduce the cost of checking density-connectedness.

Through an extensive evaluation carried out under various configurations, it is demonstrated that $\mathcal{DISC}$ is highly effective especially when clusters need to be updated frequently with a small stride. In most practical settings, $\mathcal{DISC}$ outperformed all the *exact* clustering methods in comparison. For detecting clusters of high resolution, $\mathcal{DISC}$ outperformed significantly all the *approximate* clustering methods in comparison in both speed and quality.

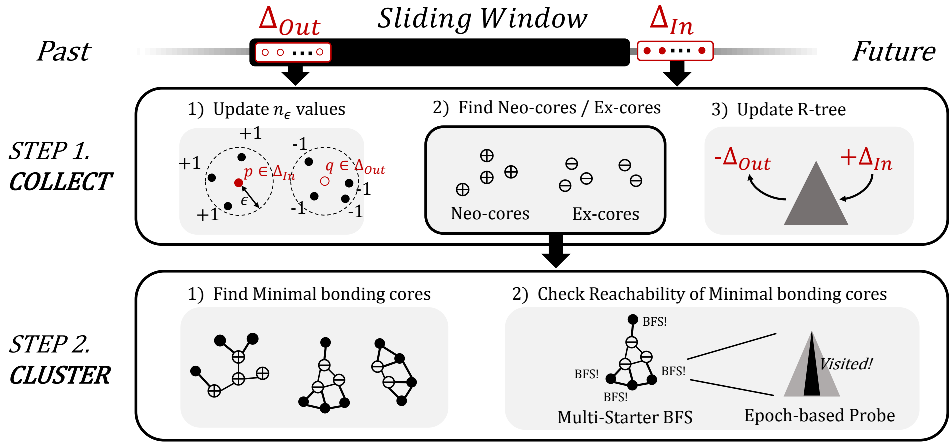


Figure 4.1: Overview of $\mathcal{DISC}$

4.1 The $\mathcal{DISC}$ Algorithm

First, an overview of the algorithm is provided, and then its two primary steps, COLLECT and CLUSTER, are described.

4.1.1 Overview of $\mathcal{DISC}$

$\mathcal{DISC}$ is no different from DBSCAN in that it assigns each individual data point to one of the three categories, *core*, *border*, and *noise*. Besides, by the time clustering is completed, a cluster id (or cid in short) will have been assigned to every data point except for those in the *noise* category.

Let $N_\epsilon(p)$ denote a set of data points within the threshold distance ϵ from p. The cardinality of $N_\epsilon(p)$, denoted by $n_\epsilon(p)$, is maintained up-to-date for each data point p. Whenever the sliding window advances by a stride, new data points may enter the window while some of the existing ones may leave it. $\mathcal{DISC}$ will then take the changes in the data population within the sliding window into account and will bring the clusters up to date by updating the $n_\epsilon(p)$ value and the category label, denoted by $l(p)$, of each point p in the current window. (Refer to Table 4.1 for more symbols adopted to describe $\mathcal{DISC}$ in this chapter.)

Symbol	Description
ϵ	distance threshold
τ	density threshold
W_{curr}	points in the current window
W_{prev}	points in the previous window
Δ_{in}	points entering the window ($W_{curr} - W_{prev}$)
Δ_{out}	points exiting the window ($W_{prev} - W_{curr}$)
$N_\epsilon(p)$	points within ϵ distance from p
$n_\epsilon(p)$	cardinality of $N_\epsilon(p)$
$l(p)$	category label of p
$p \overset{-}{\rightsquigarrow} q$	q is *retro-reachable* from p
$p \overset{+}{\rightsquigarrow} q$	q is *nascent-reachable* from p
$\mathcal{M}^-$	minimal bonding cores for *ex-cores*
$\mathcal{M}^+$	minimal bonding cores for *neo-cores*

Table 4.1: Notations

Apparently, recomputing $n_\epsilon(p)$ and $l(p)$ values for every point p in the current window is the primary task for $\mathcal{DISC}$ to update clusters. This will be carried out in two separate steps called `COLLECT` and `CLUSTER`, which are summarized in Figure 4.1.

The `COLLECT` step updates $n_\epsilon(p)$ for every point p in the current window, and resets or initializes $l(q)$ of every point q leaving or entering the sliding window. It then identifies *ex-cores* and *neo-cores* among the points that remain in the current window, and updates a spatial R-tree index accordingly for the changes. The notions of *ex-cores* and *neo-cores* are the cornerstone of $\mathcal{DISC}$ and will be defined in Section 4.1.2.

The `CLUSTER` step finds the *minimal bonding cores* of each *ex-core* and each *neo-core*, determines the types of cluster evolution by checking reachability, and finally recomputes the cluster labels for every point in the current window. The *minimal*

bonding cores are the key idea that enables $\mathcal{DISC}$ to update clusters efficiently. They will be defined in Section 4.1.3.

4.1.2 COLLECT

When a point p enters or exits the sliding window, it changes the number of ϵ-$neighbors$ for all the ϵ-$neighbors$ of p. In other words, for any point $q \in N_\epsilon(p)$ in the current window, the $n_\epsilon(q)$ value needs to be updated. Let Δ_{out} and Δ_{in} denote the set of points exiting the window and the set of points entering the window, respectively. Then, for any point $q \in N_\epsilon(p)$, $n_\epsilon(q)$ will decrease if $p \in \Delta_{out}$ (Line 6 of Algorithm 1), and it will increase if $p \in \Delta_{in}$ (Line 12). At the end of the `COLLECT` step, every data point in the current window will have an up-to-date n_ϵ value.

Algorithm 1: COLLECT (Δ_{in}, Δ_{out})

1 $C_{out} \leftarrow \varnothing$; // C_{out} : `ex-cores in` Δ_{out}
2 **foreach** $p \in \Delta_{out}$ **do**
3 **if** $l(p) = core$ **then** $C_{out} \leftarrow C_{out} \cup \{p\}$
4 **else** delete p from the R-tree index
5 **foreach** $q \in N_\epsilon(p)$ **do**
6 **if** $l(q) \neq deleted$ **then** $n_\epsilon(q)$--
7 $l(p) \leftarrow deleted$, $n_\epsilon(p) \leftarrow 0$
8 **foreach** $p \in \Delta_{in}$ **do**
9 Insert p into the R-tree index
10 $l(p) \leftarrow unclassified$, $n_\epsilon(p) \leftarrow 1$
11 **foreach** $q \in N_\epsilon(p)$ **do**
12 **if** $l(q) \neq deleted$ **then** $n_\epsilon(q)$++, $n_\epsilon(p)$++
13 Compute the sets {*ex-cores*} and {*neo-cores*}
14 **return** ({*ex-cores*}, {*neo-cores*}, C_{out})

Another major work to be done in this step is to identify a set of *ex-cores* and a set of *neo-cores* defined below. Let W_{curr} denote the set of points in the current window, and let W_{prev} denote the set of points in the previous window.

Definition 6. (Ex-core) *A data point p that was a core in the previous window is called an* ex-core *if it already exited the current window (*i.e.*, $p \in \Delta_{out}$) or it is still in the current window but no longer a core (*i.e.*, $p \in W_{prev} \cap W_{curr}$).* $\triangleleft$

Definition 7. (Neo-core) *A data point that is a core in the current window is called a* neo-core *if it just entered the current window (*i.e.*, $p \in \Delta_{in}$) or it was not a core in the previous window (*i.e.*, $p \in W_{prev} \cap W_{curr}$).* $\triangleleft$

In the next `CLUSTER` step, these two mutually exclusive sets of *ex-cores* and *neo-cores* will play a critical role in determining the types of cluster evolution such as *split* and *merger* among others.

Note that the `COLLECT` algorithm uses an R-tree index to facilitate the retrieval of ϵ-$neighbors$ of a given point. Obviously, whenever the sliding window advances, it has to maintain the R-tree index up to date by adding and removing entries as data points enter and leave the window. However, the *ex-cores* in Δ_{out} will not be removed from the R-tree index until both the `COLLECT` and `CLUSTER` steps are completed. This is because the `CLUSTER` step will have to access *ex-cores* in Δ_{out} as well as those in $W_{prev} \cap W_{curr}$. For the reason, all the *ex-cores* that exited the window are collected (Line 3 of Algorithm 1) and passed to the `CLUSTER` step in a set denoted by C_{out}. Note that the set C_{out} is equivalent to $\{$*ex-cores*$\} \cap \Delta_{out}$.

4.1.3 CLUSTER

The *ex-cores* and *neo-cores* defined in the previous section determine collectively whether a cluster should be split and whether clusters should be merged. Besides, the

Algorithm 2: CLUSTER (*ex-cores*, *neo-cores*, C_{out})

```
// {ex-cores}, {neo-cores}, C_out from COLLECT
1  E ← {ex-cores}
2  while E ≠ ∅ do
3      Compute R⁻(p) and M⁻(p) for p ∈ E
4      ncc ← MS-BFS(M⁻(p))
               // ncc: # of connected components in M⁻(p)
5      if ncc > 1 then a cluster splits
6      else a cluster shrinks or dissipates
7      E ← E − R⁻(p)                   // Avoid redundant work
8  Remove C_out from the R-tree index
9  N ← {neo-cores}
10 foreach p ∈ N do
11     if M⁺(p) is disconnected then clusters merge
12     else a cluster grows or emerges
13     N ← N − R⁺(p)                   // Avoid redundant work
```

other types of cluster evolution such as *emergence*, *dissipation*, *expansion*, and *shrink* can also be determined solely by the *ex-cores* and *neo-cores*.

The CLUSTER step presented in this section provides a sophisticated but highly efficient procedure to expedite the processing of cluster evolution. The high-level description of the procedure is given in Algorithm 2. As can be seen in the pseudocode, *ex-cores* are used to process splitting clusters while *neo-cores* are used to process merging clusters. Between these two main operations, splitting a cluster is computationally much more intensive for updating clusters incrementally. Each of the sub-procedures of the algorithm will be described in detail in this section.

4.1.3.1 Splitting a Cluster

A cluster split involves a breakup of density-reachability between core points. When a core point loses its status to become an *ex-core*, it may cut a density-reachable path between the cores in the same cluster, which in turn may contribute to a cluster split event. Essentially, a cluster can only be split when *ex-cores* break up a density-reachable path between two core points in the same cluster and there is no more path left between them.

In fact, splitting a cluster can be expedited by consolidating all the *ex-cores* turned up when the sliding window advances. A considerable amount of redundant work can be avoided by checking the density connectedness only for the *minimal bonding cores* (that will be defined below) and by minimizing the number of range searches required.

In an attempt to clearly specify the minimal set of cores to examine, the notions of *retro-reachability* (Definition 8) and *minimal bonding cores* of an *ex-core* (Definition 9) were defined below. (See Figure 4.2 for illustration.)

Definition 8. *For a pair of* ex-cores p *and* q, p *is* directly retro-reachable *to* q *if* p *was directly density-reachable to* q *with respect to the previous window* W_{prev}. *More generally,* p *is* retro-reachable *to* q *(denoted by* $p \overset{-}{\rightsquigarrow} q$*) if there is a chain of* directly retro-reachable ex-cores *from* p *to* q. $\triangleleft$

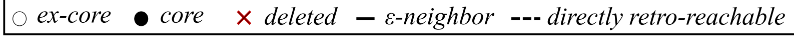
Unlike density-reachability, the retro-reachability is transitive and symmetric because this relation is defined for *ex-cores* only. That is, $p \overset{-}{\rightsquigarrow} q$ is equivalent to $q \overset{-}{\rightsquigarrow} p$.

For an *ex-core* p, let $\mathcal{R}^-(p)$ denote a set of *ex-cores* that are retro-reachable from p. Formally,

$$\mathcal{R}^-(p) = \{q \in W_{prev} \mid p \overset{-}{\rightsquigarrow} q\}.$$

Note that $p \in \mathcal{R}^-(p)$ and retro-reachability is reflexive.

Now the *minimal bonding cores* of an *ex-core* is defined, which will be used to pose a necessary condition for the *ex-core* to trigger splitting a cluster. Note that not every *ex-core* necessarily splits a cluster.



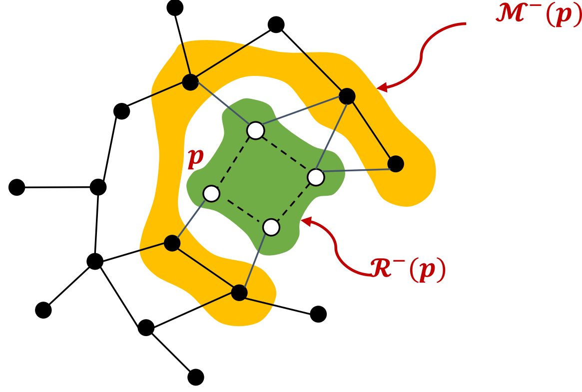


Figure 4.2: Minimal bonding cores of an *ex-core* p

Definition 9. *For an* ex-core p*, the set* $\mathcal{M}^-(p)$ *of its* minimal bonding cores *is defined to be*

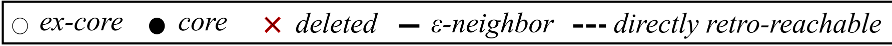
$$\mathcal{M}^-(p) = \{q \mid (q \textit{ is a core in both } W_{prev} \textit{ and } W_{curr}) \\ \wedge\ (q \in N_\epsilon(r) \textit{ for some } r \in \mathcal{R}^-(p))\}$$

◁

The second half of the condition, namely "$q \in N_\epsilon(r)$ for some $r \in \mathcal{R}^-(p)$" requires that q must be among the ϵ-$neighbors$ of a certain *ex-core* that is retro-reachable from p. It is this condition which the minimality of $\mathcal{M}^-(p)$ comes from. Among the cores that are density-reachable to the *ex-cores* in $\mathcal{R}^-(p)$, only those *directly* density-reachable to some *ex-core* are included in $\mathcal{M}^-(p)$.

For example, in Figure 4.3, border P_1 and core P_2 are about to exit the current window. Exiting P_1 turns its adjacent cores, B and K, to *ex-cores*. Exiting P_2 also turns its adjacent cores, D and F, as well as itself to *ex-cores*. The set of *ex-cores* that are retro-reachable from B is $\mathcal{R}^-(B) = \{B, D, F, K, P_2\}$, and the *minimal bonding cores* of B is $\mathcal{M}^-(B) = \{A, C, E, G, H, J\}$.

Combined with the minimality of $\mathcal{M}^-(p)$, the following lemmas and theorem al-



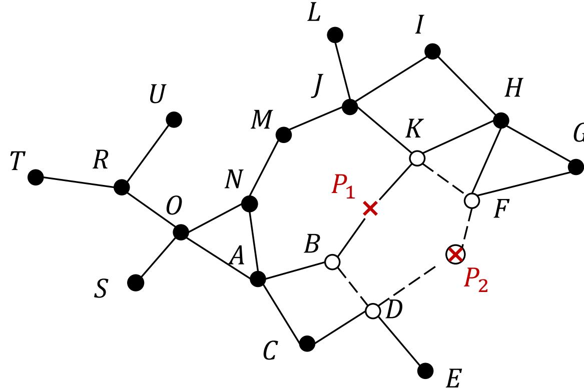


Figure 4.3: Cluster evolution by sliding window

low us to focus on the minimal set of core points when determining whether any cluster would be split by an *ex-core* p or any of its retro-reachable *ex-cores*.

Lemma 1. *For* ex-cores *p and q, $\mathcal{M}^-(p) = \mathcal{M}^-(q)$ if $\mathcal{R}^-(p) = \mathcal{R}^-(q)$.*

Proof. For $\forall x \in \mathcal{M}^-(p)$, x was and is a core in W_{prev} and W_{curr} such that $x \in N_\epsilon(r)$ for some $r \in \mathcal{R}^-(p)$. If $\mathcal{R}^-(p) = \mathcal{R}^-(q)$, then it holds that $x \in \mathcal{M}^-(q)$. Therefore, $\mathcal{M}^-(p) \subseteq \mathcal{M}^-(q)$. It can be shown that $\mathcal{M}^-(q) \subseteq \mathcal{M}^-(p)$ in the same way. □

Lemma 2. *An* ex-core *p does not split the cluster it belongs to if $\mathcal{M}^-(p)$ is* density-connected.

Proof. (By contradiction.) Suppose a cluster $\mathcal{C}$ containing p in the previous window is being split to two non-empty separate clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. Any point $x \in \mathcal{C}_1$ was density-reachable to p in the previous window because both x and p were in $\mathcal{C}$. So there must exist $x' \in \mathcal{C}_1$ that was on the density-reachable path and closest to p. This implies that x' is an ϵ-neighbor of p or one of $\mathcal{R}^-(p)$. Thus, by definition, $x' \in \mathcal{M}^-(p)$.

Similarly, there must exist $y' \in \mathcal{C}_2$ such that $y' \in \mathcal{M}^-(p)$. Since $\mathcal{M}^-(p)$ is density-connected, x' and y' are density-reachable from each other. This is a contraction to the assumption that $\mathcal{C}_1$ and $\mathcal{C}_2$ are two separate clusters. □

Theorem 1. *For an* ex-core p*, if* $\mathcal{M}^-(p)$ *is* density-connected*, none of the* ex-cores *in* $\mathcal{R}^-(p)$ *splits a cluster.*

Proof. For any *ex-core* $x \in \mathcal{R}^-(p)$, $\mathcal{R}^-(x) = \mathcal{R}^-(p)$ because the retro-reachability is symmetric and transitive. Then, it follows that $\mathcal{M}^-(x) = \mathcal{M}^-(p)$ by Lemma 1. Therefore, by Lemma 2, x does not split the cluster it belonged to in the previous window. □

The implication of Theorem 1 is that examining any one of the *ex-cores* in $\mathcal{R}^-(p)$ will obviate the need for all the other *ex-cores* in $\mathcal{R}^-(p)$ (Line 7 of Algorithm 2). This will let us avoid redundant work and reduce the number of range searches significantly.

Now let us turn our attention to cluster evolution caused by *ex-cores*. For each *ex-core* p, all *ex-cores* in $\mathcal{R}^-(p)$ can be discovered by executing $|\mathcal{R}^-(p)|$ range searches. Since all the cores in $\mathcal{M}^-(p)$ are ϵ-$neighbors$ of an *ex-core* in $\mathcal{R}^-(p)$, they will also be discovered with no additional search. Once the set $\mathcal{M}^-(p)$ of minimal bonding cores is computed for each *ex-core* p, we are ready to determine the types of cluster evolution caused by them. If $\mathcal{M}^-(p)$ is not density-connected (*i.e.*, there is more than one connected component), then the cluster from the previous window will be split in the current window (Line 5 of Algorithm 2). If $\mathcal{M}^-(p)$ is density-connected, the cluster will be simply $shrunk$ in size (Line 6). If $\mathcal{M}^-(p)$ is empty, then the cluster will be $dissipated$ completely.

Checking the connectedness of $\mathcal{M}^-(p)$ can be done by a BFS traversal, which requires executing a number of range searches against the R-tree index. When $\mathcal{M}^-(p)$ is large, this overhead may become significant and warrant careful coordination and optimization. In order to do it efficiently, a variant of breadth-first search called *Multi-Starter BFS* (invoked in Line 4 of Algorithm 2) and an *Epoch-Based* probing method

for the R-tree index are proposed. The number of range searches to execute can be considerably reduced by the former, and the individual range searches can be performed more quickly by the latter. They will be described in detail in Section 4.2.

The following examples illustrate how DBSCAN and the proposed $\mathcal{DISC}$ method deal with cluster evolution caused by *ex-cores* and compare these methods with respect to the minimum number of range searches required by each method.

Example 2. *Consider the evolving cluster shown in Figure 4.3. DBSCAN performs the clustering procedure from scratch. Specifically, when each of P_1 and P_2 is excluded from the window, a BFS traversal is performed for every point in the window. At least 19 range searches are required, and the number of range searches would be higher if $noise$ and $border$ points were taken into account.* $\diamondsuit$

Example 3. *Consider again the same scenario given in Example 2. After the exclusion of P_1 and P_2, there are five* ex-cores*, $B, D, F, K,$ and P_2, which turned up in the current window. The* `CLUSTER` *algorithm of $\mathcal{DISC}$ finds $\mathcal{R}^-(p)$ and $\mathcal{M}^-(p)$ for each $p \in \{B, D, F, K, P_2\}$ by executing five range searches. Then, a BFS traversal is performed for each of the five minimal bonding core sets. Although more work appears to be required with an increased number of BFS traversals, the opposite is true. This is because all the five* ex-cores *are retro-reachable from one another, and hence*

$$\mathcal{R}^-(B) = \mathcal{R}^-(D) = \mathcal{R}^-(F) = \mathcal{R}^-(K) = \mathcal{R}^-(P_2).$$

Thus, once $\mathcal{M}^-(B)$ is processed and all the ex-cores *in $\mathcal{R}^-(B)$ are excluded from further consideration (by Line 7 of Algorithm 2 that will make the set E empty in this case), there will be no more* ex-cores *left to process the minimal bonding core sets for. A BFS traversal will only be performed for $\mathcal{M}^-(B) = \{A, C, E, G, H, J\}$ by executing no more than six range searches. Therefore, the minimum number of range searches is reduced further down to eleven.* $\diamondsuit$

4.1.3.2 Merging Clusters

After all *ex-cores* are processed, the `CLUSTER` algorithm starts examining *neo-cores* to see whether existing clusters would have to be merged (Lines 9-13 of Algorithm 2). Clusters are merged when cores from different clusters become density-reachable. Since only a *neo-core* can contribute to creating a new density-reachable path, existing clusters can be merged only when an existing point gains the core status or a new core enters the current window.

Much the similar way done for *ex-cores*, the *nascent-reachability* (Definition 10) and the *minimal bonding cores* of a *neo-core* (Definition 11) are defined below.

Definition 10. *For a pair of* neo-cores p *and* q*,* p *is* directly nascent-reachable *to* q *if* p *is directly density-reachable to* q *with respect to the current window* W_{curr}*. In general,* p *is* nascent-reachable *to* q *(denoted by* $p \overset{+}{\rightsquigarrow} q$*) if there is a chain of* directly nascent-reachable neo-cores *from* p *to* q*.* ◁

For a *neo-core* p, let $\mathcal{R}^+(p)$ denote a set of *neo-cores* that are nascent-reachable from p. Formally,

$$\mathcal{R}^+(p) = \{q \in W_{curr} \mid p \overset{+}{\rightsquigarrow} q\}.$$

Like the retro-reachability, the nascent-reachability is reflexive, symmetric and transitive. Therefore, $p \overset{+}{\rightsquigarrow} q$ is equivalent to $q \overset{+}{\rightsquigarrow} p$, and $p \in \mathcal{R}^+(p)$.

Definition 11. *For a* neo-core p*, the set* $\mathcal{M}^+(p)$ *of its* minimal bonding cores *is defined to be*

$$\begin{aligned}\mathcal{M}^+(p) = \{q \mid\ & (q \text{ is a core in both } W_{prev} \text{ and } W_{curr}) \\ & \wedge\ (q \in N_\epsilon(r) \text{ for some } r \in \mathcal{R}^+(p))\}\end{aligned}$$

◁

The minimal bonding cores of *neo-cores* defined above enable us to determine the types of cluster evolution caused by them. For a *neo-core* p, if $\mathcal{M}^+(p)$ is empty, then a

new cluster $emerges$, which consists solely of the *neo-cores* in $\mathcal{R}^+(p)$. If all the cores in $\mathcal{M}^+(p)$ belong to one cluster, then all the *neo-cores* in $\mathcal{R}^+(p)$ are added to that cluster and let it grow in size (*expansion*). If the cores in $\mathcal{M}^+(p)$ are spread over more one cluster, say $\mathcal{C}_1, \ldots, \mathcal{C}_k$, then all the cores in $\mathcal{C}_1, \ldots, \mathcal{C}_k$ are merged into a single cluster together with the *neo-cores* in $\mathcal{R}^+(p)$.

Despite all the similarities between $\mathcal{M}^-(q)$ of an *ex-core* q and $\mathcal{M}^+(p)$ of a *neo-core* p, there is a striking difference between them. While the connectedness of the cores in $\mathcal{M}^-(q)$ must be checked for each *ex-core* q by executing a number of range searches (Line 4 of Algorithm 2), it is not necessary to do that for the cores in $\mathcal{M}^+(p)$ of any *neo-core* p. We have only to find out whether $\mathcal{M}^+(p)$ is empty or how many clusters the cores in $\mathcal{M}^+(p)$ are spread over (Line 11), which can be done quickly just by examining the labels of the cores. Therefore, the cluster evolution caused by *neo-cores* will be handled with much more ease than the cluster evolution caused by *ex-cores*.

4.1.4 Horizontal Manner vs. Vertical Manner

The basic $\mathcal{DISC}$ algorithm is run in two steps, COLLECT and CLUSTER which process ΔD_{Out} and ΔD_{In} together (Horizontal manner). But it can be run in another two steps, (1) processing ΔD_{Out} first and (2) processing ΔD_{In} next (Vertical manner). Therefore, $\mathcal{DISC}$ is not restricted to the sliding window model. Each operation can be used independently when either batch deletion or batch insertion is required.

However, it is worth noting that the horizontal manner would be more advantageous than the vertical manner in the sliding window model since it can prune a redundant change and lower the computational overhead.

For example, when ΔD_{Out} and ΔD_{In} are processed in serial order, the state ($core$, $border$ or $noise$) of each point changes as following cases. C denotes $core$, B denotes $border$, and N denotes $noise$. The first arrow denotes the state change due to the deletion, and the second arrow denotes the state change due to the insertion.

1. $B \backslash N \rightarrow B \backslash N \rightarrow C$: When the $border$ or $noise$ points do not change their states after the deletion of ΔD_{Out}, but change to $core$ points after the insertion of ΔD_{In}, they are considered as newly created core points, so these points should be processed.

2. $C \rightarrow B \backslash N \rightarrow B \backslash N$: Points which change in this scenario also affect the evolution of the clusters. Therefore, they should be processed.

3. $C \rightarrow B \backslash N \rightarrow C$: If $\mathcal{DISC}$ processes deletion and insertion separately with the bulk deletion and bulk insertion, it unnecessarily processes this kind of data points, because changing core property of each point affects the evolution of the cluster. Therefore they have to be processed twice. However, by considering deletion and insertion together with the horizontal manner, these data points can be pruned, since their states do not change eventually. In an extreme situation, we can expect to save lots of time when most of the points are belong to this scenario. This is why we divide the overall process into the horizontal manner.

4.2 Checking Reachability

Whether a cluster is split by an *ex-core* is determined by the density-reachability among the minimal bonding cores of the *ex-core*. For a given pair of cores, the density-reachability can be checked by executing a series of range searches against the R-tree index starting from either core. Only when the search encounters the other core before exhausting all reachable cores, the pair will be declared density-reachable. This procedure is essentially a variant of breadth-first search (BFS) commonly used for graph traversal. Considering the potentially high cost of reachability checks requiring a number of range searches, *Multi-Starter BFS* and *Epoch-Based* probing strategy for the R-tree index are proposed.

Note that range searches against the R-tree index could be avoided entirely if the ϵ*-neighbor* relations between cores were materialized in a graph. Then the reachability

checks could be done more quickly by traversing the materialized graph. However, it is not chosen because the $\mathcal{O}(n^2)$ cost of maintaining a materialized graph can be too high with n being the number of cores in the graph.

4.2.1 Multi-Starter BFS

In order to check the density-connectedness of $\mathcal{M}^-(p)$ for an *ex-core* p efficiently, a new search procedure called *Multi-Starter Breadth-First Search (MS-BFS)* has been developed. This is an extension of the traditional breadth-first search.

Algorithm 3: *MS-BFS* ($\mathcal{M}^-(p)$)

```
1  ncc ← 0                          // # of connected components
2  M ← M⁻(p)
3  Q_{s∈M} ← EmptyQueue
4  foreach s ∈ M do  Q_s.enque(s)
5  while |M| > 1 do
       // Run BFS_s for each s ∈ M simultaneously
6      if Q_s is empty then  ncc++, M ← M − {s}
7      else
8          r ← Q_s.deque
9          foreach core x ∈ N_ε(r) unvisited by BFS_s do
10             if x is visited by BFS_t then
11                 Q_s ← Q_s ∪ Q_t, M ← M − {t}
12             else  Q_s.enque(x)
13 return ncc
```

Imagine a (non-materialized) graph $G(V, E)$ whose vertex set V consists of core points and whose edge set E consists of pairs of cores that are ϵ-$neighbors$ to each other. The *MS-BFS* initiates a breadth-first search starting from each vertex in $\mathcal{M}^-(p)$

of G simultaneously. When two searches meet at a certain vertex, they merge their queues of vertices into one and restart as a single search with the merged queue (Line 11 of Algorithm 3). If all those searches are combined into one, then the graph G is connected, which indicates that all the cores in $\mathcal{M}^{-}(p)$ are density-connected. Otherwise, the graph G has more than one connected component and $\mathcal{M}^{-}(p)$ is not density-connected. Specifically, as shown in Line 6, if one of the queues becomes empty before all the vertices in G are visited, that thread of the *MS-BFS* terminates with its own connected component. In this case, the connected component does not cover the entire set of vertices in G. Thus the graph G is not connected, and neither is the set $\mathcal{M}^{-}(p)$ of minimal bonding cores. That is, a cluster split happens. Even in the case of $split$, *MS-BFS* does not explore the entire cluster. Instead, it terminates BFS procedure when there lefts only one queue, which reduces the scope of exploration.

It should be noted that the *MS-BFS* presented in this chapter is completely different from Then *et al.* 's Multi-Source BFS [79]. The Multi-Source BFS executes multiple *independent* BFS traversals over the same graph simultaneously and focuses on reducing the memory accesses when every vertex is visited multiple times. On the other hand, *MS-BFS* aims at reducing the scope of exploration by starting BFSs from multiple starters concurrently thereby reducing the number of range searches made against the R-tree index.

4.2.2 Epoch-Based Probing of R-tree Index

In the conventional BFS graph traversal, an array of Boolean flags is used to separate visited vertices from unvisited ones. Such an array of Boolean flags, however, does not help us reduce the cost of checking density-reachability because those flags will be referenced only after the ϵ-$neighbors$ of a certain core are identified by a complete range search. Consequently, the cost of avoiding an already visited core (as much as visiting an unvisited one) would remain as high as $\Omega(d)$, where d is the depth of the R-tree index.

An easy fix to this problem is to store the Boolean flags in the R-tree index itself instead of a separate array. If an entry in a leaf node is marked as visited, then the corresponding core will be ignored. If an entry in an internal node is marked as visited, then all the cores indexed in the subtree rooted at the entry will be ignored altogether. Unfortunately, however, this approach introduces another problem. Whenever another density-reachability checking *MS-BFS* is initiated, all the Boolean flags of the R-tree index must be reset beforehand, and this overhead may not be trivial.

Algorithm 4: $Epoch_Based_Probe(range, node, tick)$

```
1 foreach entry in node do
2     if range covers entry and entry.epoch < tick then
3         if node is a leaf then entry.epoch ← tick
4         else Epoch_Based_Probe(range, entry, tick)
5 node.epoch ← min(entries.epoch)
```

This concern is addressed by adopting an epoch-based method that stores *epochs of a visiting history* rather than just Boolean visited-or-not flags in the R-tree index. This method can be implemented efficiently with a monotonically increasing counter. When a density-reachability checking *MS-BFS* begins anew, a *tick* value is assigned from the counter so that each individual *MS-BFS* instance is given a distinct tick value.

An entry in a leaf node takes the current *tick* value as its *epoch* when the entry (and its core) is visited (Line 3 of Algorithm 4). Thus, an epoch value smaller than the current tick implies that the core referenced by the leaf entry has not been visited by the current instance of *MS-BFS*. On the backtracking, the range search adjusts the epoch of a parent entry such that it is always equal to the *minimum* of all epochs in its child entries (Lines 5). Thus, the epoch of an internal entry smaller than the current tick implies that there exists at least one child entry that has not been visited by the current instance of *MS-BFS*. The checking procedure can ignore a core or a group of

cores altogether if an index entry encountered has an epoch equal to the current tick.

Note that even with this epoch-based method, the cost of finding unvisited ϵ-$neighbors$ will remain as $\Omega(d)$. Nonetheless, this method can reduce the cost of probing the R-tree index quite considerably by pruning out any unnecessary portion of the index each range search has to probe.

4.3 Updating Labels

The ultimate goal of $\mathcal{DISC}$ is to label each core or border point in the current window correctly with the id of the cluster (or *cid*) it belongs to. Since the cluster membership of points may change as the sliding window advances, the `CLUSTER` algorithm of $\mathcal{DISC}$ handles it by updating the labels for *ex-cores*, *neo-cores* and any point which is affected by *ex-cores* and *neo-cores*.

The labels of *ex-cores* may change to $border$ or $noise$, and the labels of $cores$ affected by *ex-cores* may change to a different cid due to the splits caused by *ex-cores*. These labels of *ex-cores* and $cores$ are updated when a $\mathcal{M}^-$ set is processed by the *MS-BFS* procedure. Similarly, the labels of *neo-cores* and $cores$ affected by them are updated so that they have the same cid when a $\mathcal{M}^+$ set is processed. Besides, non-core points near *ex-cores* and *neo-cores* can also change their labels. Labels of these points are instantly updated if they are visited while *minimal bonding cores* are processed. Otherwise, they will be updated later by examining labels of their ϵ-$neighbors$. Eventually, all the $core$ and $border$ points of the same connected component will share the same cid, and this guarantees a set of clusters identical to what DBSCAN would produce.

The $\mathcal{DISC}$ algorithm covered in this chapter produces the same clustering results as DBSCAN. By minimizing the computational burden with batch operations, $\mathcal{DISC}$ achieves faster speed than existing approaches. Nonetheless, $\mathcal{DISC}$ has the same time complexity as Incremental DBSCAN, and it still suffers from the slow deletion problem. In the following chapter, the problem will be addressed squarely, and a novel approach will be proposed to address it.

Chapter 5

Avoiding Graph Traversals in Updating Clusters

Deleting a point requires a costly graph traversal to check whether the cluster is still connected after the deletion. $\mathcal{DISC}$ alleviates the performance degradation but does not fundamentally address the problem. The density-based clustering method presented in this chapter is called *DenForest*. It produces similar (but not exactly the same) density-based clusters to DBSCAN. It addresses the slow deletion squarely and shows very efficient performance in the deletion process. The time and space complexity of the previous algorithms as well as *DenForest* are summarized in Table 5.1. The Incremental DBSCAN and ρ-double-approximate DBSCAN algorithms are referred to as IncDBSCAN and ρ_2-Approx, respectively.

As an incremental density-based clustering algorithm, *DenForest* is based on a novel idea that allows us to manage clusters as a group of *spanning trees* of data points rather than a graph. In general, it is far simpler to determine whether the removal of a point splits a tree than a graph. However, a spanning tree being split does not always imply the underlying graph is also split. Therefore, a new data structure called *DenTree* is designed, which can tell accurately whether the underlying graph is being split or not. The *DenTrees* of a graph can determine all by themselves whether the removal of a point splits the graph (*i.e.*, clusters). By managing clusters as *DenTrees*, *DenForest* addresses the slow deletion problem and achieves fast incremental density-

Table 5.1: Time and Space Complexity

Algorithm	Deletion	Insertion	Space
IncDBSCAN [27]	$O(N(N^{1-1/d}+k))$	$O(kN^{1-1/d}+k^2)$	$O(N)$
ExtraN [87]	-	$O(N^{1-1/d}+k(\frac{\lvert W\rvert}{\lvert S\rvert})^2)$	$O(N\frac{\lvert W\rvert}{\lvert S\rvert})$
ρ_2-*Approx* [34]	$O(log^2N)$	$O(log^2N)$	$O(NlogN)$
DISC	$O(N(N^{1-1/d}+k))$	$O(kN^{1-1/d}+k^2)$	$O(N)$
DenForest	$\boldsymbol{O(logN)}$	$\boldsymbol{O(N^{1-1/d}+klogN)}$	$\boldsymbol{O(N)}$

N is the number of points within a sliding window, k is the number of points retrieved by a range search [9], and d is the dimensionality of data. |W| and |S| denote the size of a sliding window and its stride, respectively.

based clustering.

DenForest takes $O(logN)$ amortized time to delete a point, which is far faster than the other algorithms in comparison. Its performance is less sensitive to the dimensionality of data since it does not require range searches. *DenForest* takes $O(N^{1-1/d} + klogN)$ amortized time to insert a point. Although it is asymptotically slower than ρ_2-Approx, *DenForest* yields much higher performance in most practical settings.

It is demonstrated through extensive experiments conducted on various real-world datasets that *DenForest* outperforms the currently available clustering algorithms considerably. It is confirmed by measuring the clustering quality in widely used metrics that the clustering quality of *DenForest* is not compromised, and *DenForest* and the DBSCAN algorithm are in fact comparable with respect to the clustering quality.

5.1 The *DenForest* Algorithm

First, an overview of the algorithm is provided, and a few key ideas such as *nostalgic cores* and *DenTree* are described in detail.

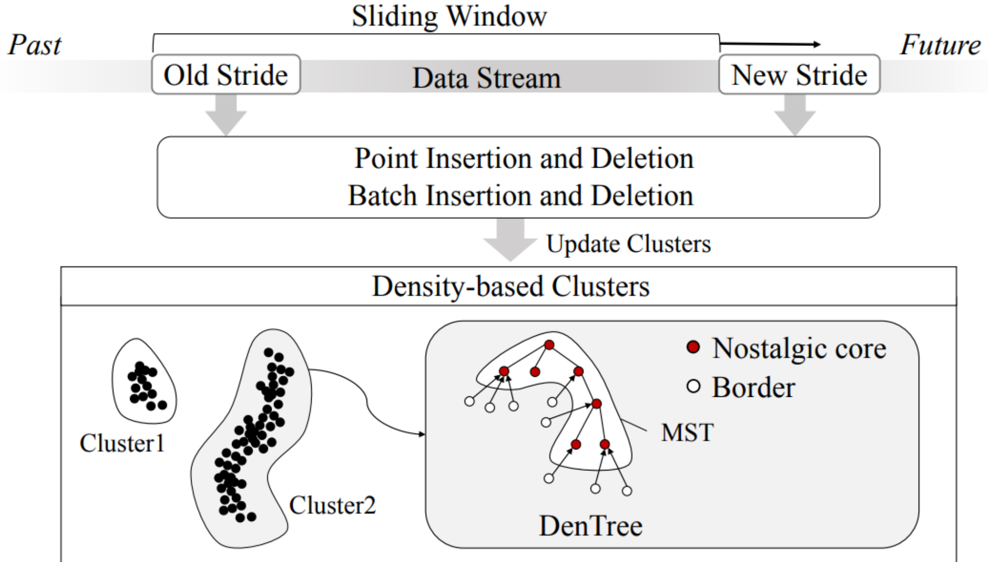


Figure 5.1: Overview of *DenForest*

5.1.1 Overview of *DenForest*

Suppose data points are generated from sources such as sensing devices and are sent over for processing continuously. Each data point is associated with a *timestamp* T that indicates the event time or the ingestion time. The sliding window covers the most recent data points in the stream at any moment in time.

For the data points in the current window, *DenForest* produces density-based clusters by detecting the connected components of *nostalgic cores* (Section 5.1.2). The *nostalgic cores* are similar to the cores defined by DBSCAN in that they are points found in the dense region. But the *nostalgic cores* differ from DBSCAN's cores in the way they expire and become non-core points. Each density-based cluster of *nostalgic cores* can be managed as a tree structure called *DenTree*, which can expedite the deletion process significantly (Section 5.1.3 and Theorem 2). When the window slides, *DenForest* updates clusters by inserting and deleting points individually (Sections 5.2.1 and 5.2.2) or in a batch (Section 5.2.5). It is assumed that data points in the

Table 5.2: Notation

Symbol	**Description**
N	the number of points in the current window
τ, ϵ	the density and distance thresholds
CC	a connected component of *cores*
$d\text{-}core$	the $core$ point defined by DBSCAN
T	the timestamp of a point
T_c	the core-expiration time of a point
$N_\epsilon(p)$	the neighboring points within ϵ-distance from p
$N'_\epsilon(p)$	the *previously inserted* points in $N_\epsilon(p)$
$\mathcal{MST}$	the maximum spanning tree of *nostalgic cores*
d	the number of dimensions
k	the number of points retrieved by a range query
M	the number of nodes in the Link-Cut tree
SC	*a super nostalgic core*
B_ϵ	a d-dimensional ball (or hypersphere) with radius ϵ
D_ϵ	the number of points in B_ϵ
$NC_\epsilon(p)$	the *nostalgic cores* within ϵ-distance from a *d-core* point p

same stride are processed together, and data points in different strides are processed strictly in the order of their timestamps. The overall clustering procedure by *DenForest* is illustrated in Figure 5.1. The denotational symbols frequently used in the chapter are summarized in Table 5.2.

5.1.1.1 Supported Types of the Sliding Window Model

DenForest supports both the count-based and the time-based sliding window models. First, under the count-based sliding window model, the stride size is one or greater.

If the stride size is one, then a point insertion and a point deletion are performed alternately. If the stride size is greater than one, then a batch insertion and a batch deletion are performed alternately. The arrival order of data points in the stream can be used as the timestamp of an individual point. Second, under the time-based sliding window model, the stride size is a fixed time duration, and a batch insertion and a batch deletion are performed alternately and periodically. The only limitation is that the window size must be a multiple of the stride size for the batch-optimized operations to be applicable.

5.1.2 Nostalgic Core and Density-based Clusters

We come to realize that the slow deletion problem is caused intrinsically by the unpredictability of a vanishing core's expiration time. In this section, a novel approach that can precisely predict the expiration time of a core when it enters the window is presented.

Similarly to the DBSCAN algorithm, *DenForest* adopts two parameters, namely density and distance thresholds (τ and ϵ) to discover density-reachable *cores* and adjacent *borders*. Unlike DBSCAN, however, *DenForest* relies on its own notion of a point being a core called a *nostalgic core* rather than that of DBSCAN. (DBSCAN's cores are referred to as *d-cores* hereinafter to distinguish one from another.) *DenForest* can determine exactly when a *nostalgic core* p will expire to become a $non\text{-}core$ point, immediately after p enters the sliding window. This is done by considering only the current data points that entered the window earlier than p.

Definition 12 (**Nostalgic core**). *A point p in the window W is a nostalgic core if the number of ϵ-neighbors of p that entered W no later than p meets the density requirement. That is, p is a nostalgic core if $|N'_\epsilon(p)| \geq \tau$ where $N'_\epsilon(p) = \{q \in W \mid q \in N_\epsilon(p) \wedge q.T \leq p.T\}$, and $p.T$ and $q.T$ denote the timestamps of p and q, respectively.*

Whether a point p is a *nostalgic core* or not is determined at the insertion time solely by the existing points in the current window, and the core status of p is not

affected by the points inserted in the future. Furthermore, when a *nostalgic core* p becomes a non-core point is also determined at the insertion time. (Refer to Lemma 3 below.) Note that DBSCAN's cores or *d-cores* do not possess any of these properties. While a point p stays in the window, DBSCAN allows p to gain or lose the core status at any time by pre-existing and future points leaving or entering the window. Note also that the set of *nostalgic cores* is always a *subset* of the set of *d-cores*.

Let $p.T_c$ denote the *core-expiration time* of p or the time when a *nostalgic core* p loses its core status to become a *non-core* point.

Lemma 3. *$p.T_c$ can be determined when p enters the window.*

Proof. Consider a point p that is about to enter the window. Assume $|N'_\epsilon(p)| \geq \tau$ and p is determined as a *nostalgic core*. Let q denote a point in $N'_\epsilon(p)$ such that its timestamp $q.T$ is the τth largest (or youngest). Then, p loses its core status when q leaves the window. Since q will leave the window at time $q.T + |W|$, p will become a non-core point at that time. That is, $p.T_c = q.T + |W|$. Therefore, the core-expiration time of p can be determined right at the moment when it enters the window. □

Lemma 4. *Once a point is not determined as a nostalgic core, then it can never become a nostalgic core until it leaves the window.*

Proof. For any point p in the window, $|N'_\epsilon(p)|$ can only decrease as the window slides. Therefore, if p is not a *nostalgic core* at the insertion time, it cannot become a *nostalgic core* until it leaves the window. □

Since *DenForest* defines its own *nostalgic cores*, the definitions of its border and noise points as well as its density-based clusters need to be altered accordingly.

Definition 13 (Border and noise of *DenForest*). *A point is a border if it is not a nostalgic core but within the ϵ-distance from any nostalgic core. Otherwise, it is considered a $noise$ point.*

Definition 14 (**Density-based cluster of *DenForest***)**.** *In the graph representation (described in Section 3.1.1), a density-based cluster is defined as a connected component of nostalgic cores as well as the borders adjacent to the connected component.*

Each density-based cluster is managed as a tree called *DenTree* introduced in Section 5.1.3. It may appear that *DenForest* will produce density-based clusters of poor quality because the *nostalgic cores* are defined without considering the data points being inserted in the future. It will be demonstrated later in this chapter that *DenForest* can produce clusters of comparable quality much more efficiently. (See Sections 5.3 and 6.5.3.4 for the detailed evaluation.)

5.1.2.1 Cluster Membership of Border

Traditionally, the cluster membership of a border point is not decided deterministically. If a border point is adjacent to two or more clusters, it can join any of the clusters. This is the way DBSCAN decides the cluster membership of a border point. In contrast, a deterministic heuristic is adopted. *DenForest* attaches a border point p to a nostalgic core with the largest T_c among those in $N_\epsilon(p)$, which in turn decides the cluster membership of p. This approach is not in conflict with the definition of clusters and is in fact beneficial for performance, because deletion of the *nostalgic core* that p is attached to always results in p becoming $noise$, hence avoiding further reclassification effort.

5.1.3 *DenTree*

In order to process a point deletion efficiently, each cluster is maintained as a tree structure called *DenTree*, which can be used as an accurate barometer of a cluster split. A *DenTree* consists of a maximum spanning tree ($\mathcal{MST}$ in short) of a *DenGraph* defined below and border points associated with it.

Definition 15 (**DenGraph**)**.** *A DenGraph $G(V, E, \mathcal{W})$ is an undirected edge-weighted graph where each vertex in V corresponds to a nostalgic core in the window, each*

edge in E corresponds to a pair of vertices within the ϵ-distance from each other, and each weight in $\mathcal{W}$ is set to the smaller T_c of the two adjacent vertices, namely

$$\forall\, \overline{pq} \in E,\ w_{\overline{pq}} = \min\{p.T_c, q.T_c\}. \tag{5.1}$$

A *DenGraph* may have one or more $\mathcal{MST}s$, each of which corresponds to a connected component of the *DenGraph*. For a pair of *nostalgic cores* in the same connected component, there may exist multiple paths between them. Thus, just a path being split does not always make them disconnected in the graph. However, if the path being split is the one on the $\mathcal{MST}$ of the connected component, then the two *nostalgic cores* are no longer connected in the graph. This property of the $\mathcal{MST}s$ is the key to addressing the slow deletion problem. Hereinafter, a maximum spanning tree of a *DenGraph* is simply referred to as an $\mathcal{MST}$ for brevity.

Theorem 2. *Consider two nostalgic cores p and q in an $\mathcal{MST}$ of a DenGraph. If another nostalgic core x on the path of the $\mathcal{MST}$ between p and q becomes a non-core point and is removed from the graph, then p and q are no longer connected not only in the $\mathcal{MST}$ but also in the graph.*

Proof. (By contradiction). Consider the moment when x is about to become a non-core point and be removed from the graph. Suppose the path on the $\mathcal{MST}$ passing through x is not the only path between p and q in the graph. Then there must be another path between them, and these two paths form a cycle. Since x is the one that is about to expire, all the points in the cycle have a *core-expiration time* greater than x's. Consider now a point y directly adjacent to x on the path of the $\mathcal{MST}$. Then, the edge $\overline{xy}$ must have the smallest weight among all the edges in the cycle, because $w_{\overline{xy}} = \min\{x.T_c, y.T_c\} = x.T_c$. This implies that the edge $\overline{xy}$ must not have been chosen for the $\mathcal{MST}$, and therefore contracts the assumption that $\overline{xy}$ is in the $\mathcal{MST}$. $\square$

The implication of Theorem 2 is that when an $\mathcal{MST}$ is split by a vanishing nostalgic core, the underlying connected component (and its corresponding cluster) is also

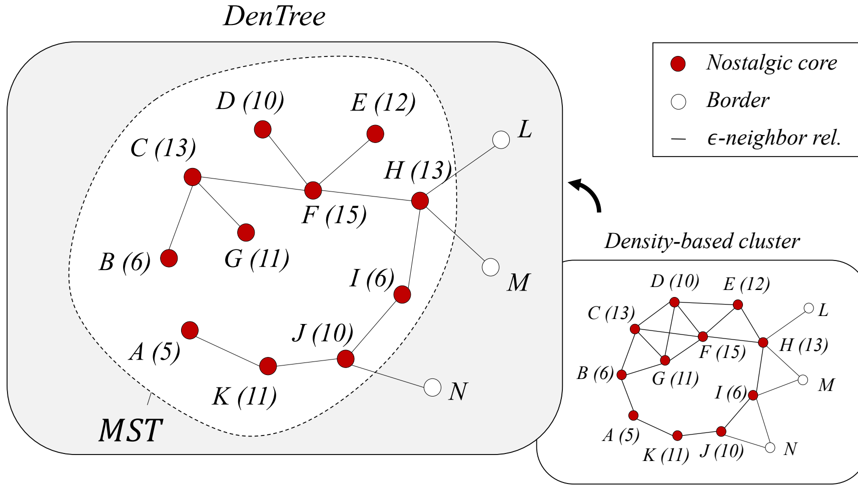


Figure 5.2: Example of *DenTree*

split. In general, a tree is split to non-empty subtrees when a node with two or more adjacent nodes is removed. Hence, to determine whether a cluster will be split by a vanishing nostalgic core p, it will be enough to check whether p is adjacent to two or more nostalgic cores in the $\mathcal{MST}$. Below *DenTree* that represents a cluster of nostalgic cores and border points is defined.

Definition 16 (***DenTree***)**.** *A DenTree is a tree composed of an $\mathcal{MST}$ and the border points associated with the $\mathcal{MST}$. If a border is adjacent to more than one nostalgic core, it is attached to the one of those that has the largest core-expiration time T_c.*

Example 4. *Figure 5.2 illustrates how a density-based cluster is represented by a DenTree, which consists of an $\mathcal{MST}$ of nostalgic cores and the border points associated with it. In the figure, the red points ($A \sim K$) and the white points ($L \sim N$) denote nostalgic cores and border points, respectively. An edge in the graph indicates that its two adjacent vertices (nostalgic cores or borders) are within the ϵ-distance from each other. Nostalgic cores are annotated with their core-expiration times. Unlike the traditional approaches, graph traversals are not required to determine whether a cluster is split or not by a vanishing core. For example, at time $t = 5$, point A becomes a non-core, and the cluster shrinks but is still connected because the DenTree is not split. On*

the other hand, at time $t = 6$, points B and I become non-cores, and the DenTree is split by point I, and consequently, the cluster is also split by point I.

5.2 Operations of *DenForest*

This section presents the detailed procedures of *DenForest*'s `Insert` and `Delete` operations. These procedures ensure that clusters produced by *DenForest* are always valid with respect to Definition 14, while the window slides.

5.2.1 Insertion

The `Insert` operation is responsible for updating clusters when new data points are added to the window. In particular, it ensures that $\mathcal{MST}s$ are updated incrementally and remain valid even when the underlying graph of *nostalgic cores* changes over time by the sliding window. The overall procedure is composed of four steps as follows. (Refer to Algorithm 5 and Figure 5.3 for details.)

STEP 1 (Point Classification) First, it determines whether a new point p is a *nostalgic core* by counting the number of its ϵ-neighbors in the current window. If the count is no less than the density threshold τ, then p is classified as a *nostalgic core*. Otherwise, it is classified as a $border$ or $noise$ point.

STEP 2 (Determination of T_c) If p is classified as a *nostalgic core*, the *core-expiration time* $p.T_c$ is computed by its ϵ-neighbors (Line 4). This step involves sorting the ϵ-neighbors in the order of their timestamps.

STEP 3 (Adding Links to $\mathcal{MST}s$) If p is a *nostalgic core*, the maximum spanning trees ($\mathcal{MST}s$) are updated by adding p and new edges adjacent to it. The point p is connected to each of the *nostalgic cores* within the ϵ-distance by an edge whose weight is set by Equation (5.1). If a cycle is formed by adding a new edge, an edge with the smallest weight is removed from the cycle by the `Connect(p,n)` function (Line 6), which is described in Algorithm 6. Three types of cluster evolution can result

Algorithm 5: Insert a point p and update the clusters

```
Insert (Point : p)
1   Insert p into the SpatialIndex
2   if |N'_ϵ(p)| ≥ τ then
3       mst ← 0 // The number of MSTs connected to p
4       p.T_c ← Core-Expiration-Time(p)
5       foreach n ∈ N'_ϵ(p) do
            // The Connect function returns true if it
               combines two disjoint MSTs.
6           if n.T_c ≥ currentTime and Connect(p, n) then
7               mst++
            end
        end
8       Determine the type of cluster evolution by the mst value
    end
9   Process the noise/borders
```

by updating the $\mathcal{MST}s$ (Line 7 and Line 8):

1. A cluster $emerges$ if there is no $\mathcal{MST}$ near p ($mst = 0$).
2. A cluster $expands$ if p is connected to one $\mathcal{MST}$ ($mst = 1$).
3. More than two clusters are $merged$ when p is connected to multiple $\mathcal{MST}s$ ($mst \geq 2$).

STEP 4 (Updating Borders) If p is a *nostalgic core*, then an existing *border* point (say x) within the ϵ-distance from p may be reconnected to p, if T_c of p is greater than that of x's adjacent *nostalgic core*. For p that is not a *nostalgic core*, if there exists a *nostalgic core* in $N_\epsilon(p)$, then p becomes a *border* point and p is connected to a *nostalgic core* in $N_\epsilon(p)$ with the largest T_c. Otherwise, p becomes a $noise$.

The following lemma proves the validity of the `Insert` operation.

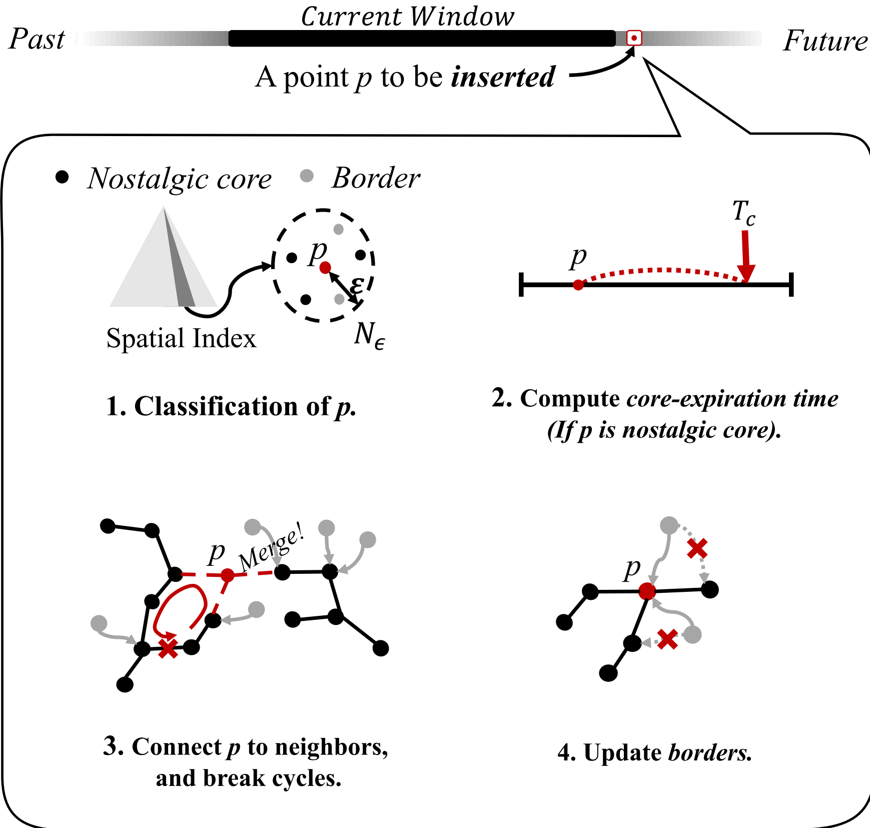


Figure 5.3: DenForest's Insertion

Lemma 5. `Insert(p)` *of Algorithm 5 updates DenTrees correctly.*

Proof. The $\mathcal{MST}s$ of a *DenGraph* remain cycle-free and maximally spanning because an edge with the smallest weight is removed if a cycle is formed by p being inserted [18]. Every border point, either a new or existing one, remains attached to a *nostalgic core* with the largest T_c within the ϵ-distance. Therefore, the *DenTrees* are updated correctly by `Insert`(p). □

Table 5.3: Link-Cut Tree Operations

APIs	Description
`Link(n,m)`	Link nodes n and m in different trees.
`Cut(n,m)`	Cut a link between nodes n and m.
`Connected(n,m)`	Check if a path exists between nodes n and m.
`FindMinE(n,m)`	Find the minimum weighted edge on the path between nodes n and m (added for *DenForest*).

5.2.1.1 $\mathcal{MST}$ based on Link-Cut Tree

DenForest relies on a data structure called *Link-Cut Tree* [76] to efficiently detect and break a cycle in the $\mathcal{MST}s$. The *Link-Cut tree* represents a set of trees and is often used to solve the dynamic connectivity problem for an acyclic graph. The trees in a *Link-Cut tree* are divided into disjoint paths, and each path is represented by a Splay tree. By managing a set of trees with a path-based structure, the *Link-Cut tree* can support its key operations in the amortized $O(\log M)$ time, where M is the total number of nodes in the trees. See Table 5.3 for the list of supported operations as well as a new one added for *DenForest*.

DenForest maintains its $\mathcal{MST}s$ in the *Link-Cut tree* and updates them whenever the underlying graph changes by a new point added to the window. For efficient updates, a new function called `FindMinE(n,m)` is designed in addition to the traditional operations of the Link-Cut tree. `FindMinE(n,m)` finds the minimum weighted edge on the path between two nodes n and m in a tree. `FindMinE(n,m)` also runs in the amortized $O(\log M)$ time.

Algorithm 6 presents the `Connect` algorithm that links two *nostalgic cores* p and q in the *Link-Cut tree*. The weight of the edge $\overline{pq}$ is set by the smaller of the *core-expiration times* of p and q (Line 1). If there already exists a path between them (Line 2), adding $\overline{pq}$ would create a cycle. Thus, the algorithm finds an edge with the

Algorithm 6: Connect two points in the Link-Cut tree

```
  Connect (Point : p, Point : q)
1   w_pq ← min{p.T_c, q.T_c}
2   if Connected(p,q) then
        // If a cycle is formed, cut the minimum weighted
         edge
3       rs ← FindMinE(p, q)
4       if w_rs ≤ w_pq then
5           Cut(r, s) and Link(p, q)
        end
6       return False // No merge
    else
7       Link(p, q)
8       return True // Potential merge
    end
```

smallest weight, say $\overline{rs}$, on the path between p and q (Line 3). If the weight of $\overline{rs}$ is smaller than the weight of $\overline{pq}$, then $\overline{rs}$ is removed and replaced by $\overline{pq}$ (Lines 4-5). Otherwise, $\overline{pq}$ is simply dropped without altering the Link-Cut tree (Line 6). If there is no path between p and q, the two separate $\mathcal{MST}s$ they belong to are linked together by adding $\overline{pq}$ (Line 7). The `Connect` algorithm returns a Boolean flag to indicate the type of cluster evolution.

5.2.1.2 Time Complexity of `Insert` Operation

The runtime of the `Insert` algorithm is given by the formula below.

$$C_i + C_r + P_{nc} \times (C_s + |N'_\epsilon|C_c) + C_p \tag{5.2}$$

C_i is the cost of inserting a point into the spatial index, C_r is the cost of a range search, P_{nc} is the probability of an inserted point being a *nostalgic core*, C_s is the cost

of sorting N'_ϵ to compute the *core-expiration time*, C_c is the cost of the `Connect` operation, and C_p is the cost of processing a $border$.

Lemma 6. *`Insert` runs in amortized $O(N^{1-1/d} + k \log N)$ time.*

Proof. Assume that the balanced *k-d tree* [9] is used as a spatial index. Then, C_i and C_r are $O(\log N)$ and $O(N^{1-1/d} + k)$, respectively, where k is the number of points retrieved by a range query. C_s and C_p are $O(|N'_\epsilon| \times \log |N'_\epsilon|)$ and $O(|N'_\epsilon|)$, respectively. C_c is amortized $O(\log M)$ because all of its sub-algorithms take amortized $O(\log M)$ time. Then, since $M < N$, the amortized time complexity of the `Insert` operation is bounded by $O(N^{1-1/d} + k \log N)$. $\square$

DenForest can work with many spatial indexes such as *R-tree* [39] and *range tree* [10] as well. The balanced *k-d tree* is assumed in the proof for its well known upperbound analysis.

5.2.2 Deletion

The `Delete` operation is responsible for updating clusters when existing data points are removed from the sliding window. Theorem 2 enables it to quickly determine whether a cluster will be split or not just by counting the links adjacent to each vanishing *nostalgic core* in the $\mathcal{MST}$. The `Delete` algorithm depicted in Algorithm 7 and Figure 5.4 runs in two main steps.

STEP 1 (Finding Expiring Nostalgic Cores) When a point q is removed from the window, some of the *nostalgic cores* may become $non\text{-}cores$. Unlike the traditional density-based methods, those expiring *nostalgic cores* can be found without executing any range search. Since the *core-expiration time* of a *nostalgic core* is determined at the insertion time and remains intact, all the *nostalgic cores* in the current window can be indexed in a supplementary data structure such as `hashmap` with their *core-expiration times* as keys. When q is removed from the window at time t, all the *nostalgic cores*

Algorithm 7: Delete a point q and update the clusters

```
Delete (Point : q)
1   E(q) : a set of nostalgic cores expired by the deletion of q
2   foreach x ∈ E(q) do
3       L ← a set of nostalgic cores linked to x
4       Determine the type of cluster evolution by the |L| value
5       foreach y ∈ L do Cut(x,y)
6       Reclassify x as either border or noise by the |L| value.
    end
7   Delete q from the SpatialIndex
```

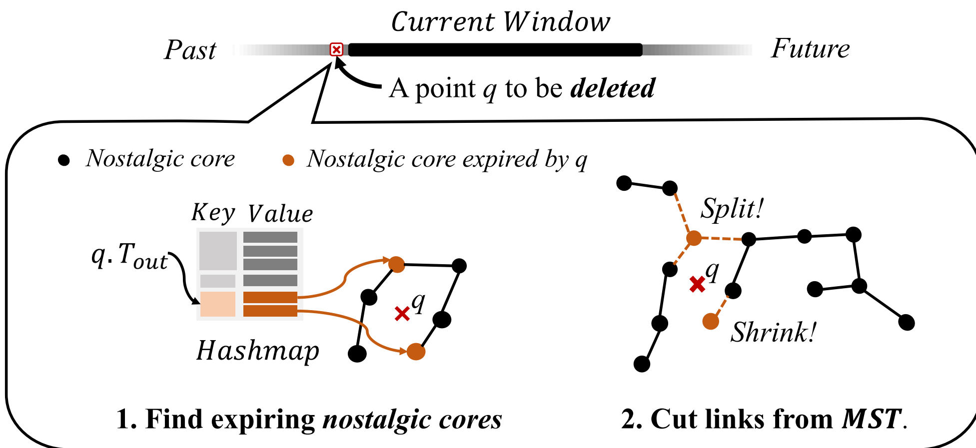


Figure 5.4: DenForest's Deletion

becoming non-cores at time t can be found in $O(|E(q)|)$ time from the supplementary data structure (Line 1).

STEP 2 (Cutting Links from $\mathcal{MST}$s) All the expiring *nostalgic cores* are examined to determine whether any $\mathcal{MST}$ is to be split. For each expiring *nostalgic core* x, all the adjacent links are found (Line 3) and removed (Line 5). Then x is reclassified as follows (Line 6). If $|L| \geq 1$, then x becomes a $border$ point and is attached to a *nostalgic core* in L with the largest T_c. If $|L| = 0$, then there is no *nostalgic core* adjacent to it and x becomes a $noise$. Three types of cluster evolution can result from each expiring *nostalgic core* x (Lines 4):

1. If $|L| = 0$, the cluster containing x dissipates.
2. If $|L| = 1$, the cluster containing x shrinks.
3. If $|L| \geq 2$, the cluster containing x is split.

Theorem 2 guarantees that the clusters updated by the `Delete` operation are valid. Note also that Algorithm 7 does not involve any range search. Consequently, the performance of the `Delete` operation is less sensitive to the dimensionality of data points, and is not overly affected by the distance threshold ϵ. This will be corroborated by the experimental evaluation in Section 6.5.3.2.

5.2.2.1 Time Complexity of `Delete` Operation

The asymptotic runtime of the `Delete` algorithm is given by the formula below.

$$O(|E| \times |L| \times \log M + \log N) \tag{5.3}$$

$|E|$ is the number of *nostalgic cores* expired by the deletion of a point, and $|L|$ is the degree (*i.e.*, the number of adjacent links) of an expiring *nostalgic core*. The terms $\log M$ and $\log N$ denote the cost of a `Cut` operation in the Link-Cut tree and the cost of a deletion in the spatial index, respectively.

The `Delete` operation can be compared with the update procedure of the $\mathcal{MST}s$ in `Insert`, where the cost of the latter is $P_{nc} \times |N'_\epsilon| \times \log M$, as shown in Equation 5.2. The term, $|E|$, can be compared with the term, P_{nc}. If the overall clusters shrink or dissipate as the window slides, the total number of *nostalgic cores* decreases. That is, P_{nc} would be smaller than $|E|$. Conversely, P_{nc} would be bigger than $|E|$ as the overall clusters emerge or expand over time.

Another term, $|L|$, can be compared with N'_ϵ. Specifically, $|L|$ cannot be larger than $|N'_\epsilon|$, since the number of links created is always smaller than the $|N'_\epsilon|$. However, the number of expiring links, $|L|$, is close to $|N'_\epsilon|$ when the clusters split a lot. Conversely, $|L|$ is close to zero when the clusters do not split often. Due to the $|N'_\epsilon|$ and the hidden constant factor in $\log M$ [1], the procedure of updating the $\mathcal{MST}$ in `Insert` is usually slower than in `Delete`. Moreover, because of the additional terms (related to range search and sorting) in `Insert`, the overall performance of `Insert` is far slower than `Delete`, which is shown in the evaluation section.

In the following theorem, the runtime of the `Delete` algorithm is bounded by determining the maximum number of *nostalgic cores* that can expire by a single point being removed from the sliding window.

Theorem 3. *The amortized runtime of the* `Delete` *algorithm is* $O(\log N)$ *where* N *is the number of points in the sliding window.*

Proof. Suppose a point p is about to be removed from the window and a *nostalgic core* x is about to become a *non-core* by that. First, x must be in $N_\epsilon(p)$. Otherwise, x would not be affected by the deletion of p. Second, the number of points that exist at exactly the same location as x must be no more than $\tau - 1$. Otherwise, x would still be a *nostalgic core* after the deletion. Third, if there are $\tau - 1$ points at the location of x, then there must be no more point within the ϵ-distance from x. Otherwise, again, x would still be a *nostalgic core* after the deletion. That is, if $\tau - 1$ *nostalgic cores* at the same location are about to expire, there must be no other point within the ϵ-distance. Hence,

[1] More LinkCut Tree's APIs are invoked in `Insert` than `Delete`

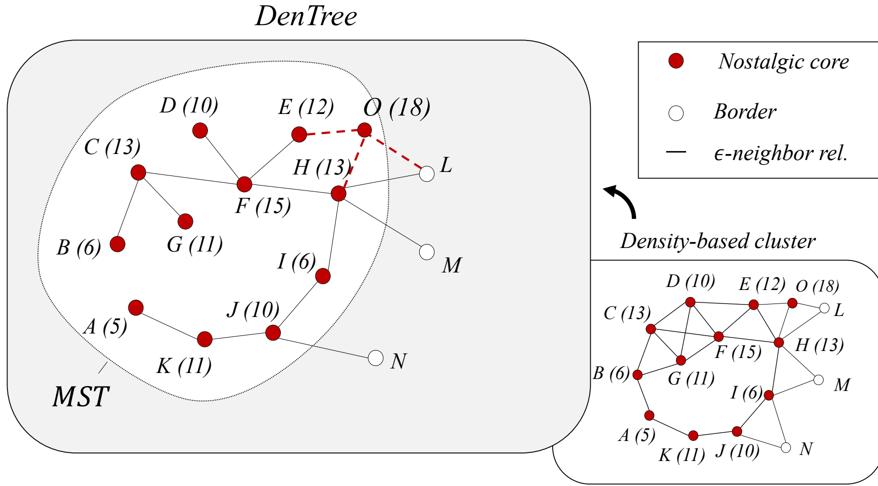


Figure 5.5: Insertion/Deletion Example

the number of those locations where groups of $\tau - 1$ expiring *nostalgic cores* coexist is bounded by a constant. In a 2-dimensional space, the maximum number of such locations is six, which is known as a *kissing number* [20].[2] In a d-dimensional space, the kissing number is bounded by c^d, where c is a small constant. Thus, the number of *nostalgic cores* expired by the deletion of p is bounded by $(\tau - 1) \times c^d$, which is $O(1)$. This implies that $|E|$ is $O(1)$ in Equation (5.3). Besides, in Equation (5.3), the average of $|L|$ is less than two (just like the average degree of a vertex in any tree or acyclic graph), and $M \leq N$ because M is the number of nodes in the Link-Cut tree. Therefore, the amortized runtime of the `Delete` algorithm is $O(\log N)$. □

Lemma 7. *DenForest consumes O(N) space.*

Proof. The main data structures *DenForest* relies on are *DenTrees*, a spatial index and a hashmap. *DenTrees* including the *Link-Cut* tree use $O(N)$ space, and the spatial index and the hashmap both use $O(N)$ space. □

[2] The kissing number is defined as the maximal number of non-overlapping unit spheres that can touch a common sphere of the same size. For *DenForest*, the radius of the spheres is $\epsilon/2$.

5.2.3 Insertion/Deletion Examples

Here are some examples for the insertion and deletion. In Figure 5.5, *DenTree* is constructed from the density-based cluster (graph). Note that only the *DenTree* is physically maintained. The graph was not physically maintained. Red points indicate the *nostalgic cores* in the window, and each edge indicates the ϵ-neighbor relationship ($i.e.$, two points are within the ϵ-distance). The numbers in parentheses indicate the core-expiration time. Only a few border points are shown in the figure for clarity of the presentation. Point O is about to be inserted.

Example 5. ***Insertion Example***

1. *Suppose that a point O is about to be inserted and is classified as a nostalgic core with a core-expiration time of 18. It has three neighbors: E, H, and L.*

2. *Because point O is a nostalgic core, edges $\overline{EO}$ and $\overline{HO}$ are connected to the MST. The weight of the edge $\overline{EO}$ is 12 (= min(12,18)), and the weight of the edge $\overline{HO}$ is 13 (= min(13, 18)).*

3. *Because cycle E-F-H-O-E occurs, the edge with the smallest weight will be removed. In this example, the two edges $\overline{EF}$ and $\overline{EO}$ have the smallest weight (= 12). Any one of the two edges is then removed from the MST to prevent the cycle.*

4. *Border L is then examined to be connected to the neighboring nostalgic core that has the larger core-expiration time. Because the core-expiration time of point O is larger than that of point H, the border point L is connected to O.*

Example 6. ***Deletion Example***

1. *As time passes, the points are deleted from the window. Because of these deleted points, some nostalgic core points become non-core points. DenForest retrieves*

these expiring nostalgic cores using a hashmap. The hashmap has the core-expiration time as a key and the corresponding nostalgic cores as values.

2. *First, nostalgic core A is retrieved at t=5 from the hashmap. (i.e., point A becomes a non-core because of the points deleted at t=5.) Then, point A is disconnected from MST. Because it is connected to only one nostalgic core, the cluster does not split.*

3. *Subsequently, two points, B and I, are retrieved at t=6. The MST is not split owing to point B, however the MST is split owing to point I. This is because point I is connected to two nostalgic cores, that is, point I causes the cluster to split.*

4. *Finally, points B and I become the borders. Point B is connected to point C. Point I is connected to point H because point H has a larger T_c than point J.*

5.2.4 Cluster Membership

DenForest does not store the cluster identification of an individual point. If it did, then the cost of updating the cluster identifications would be non-trivial. Instead, upon request, *DenForest* assigns a unique ID to the points belonging to each cluster by traversing the corresponding *DenTree*. This procedure requires $O(N)$ time, which is no worse than any existing method.

5.2.5 Batch-Optimized Update

The `Insert` and `Delete` operations can be further optimized by exploiting the locality of the data points in the same stride. By consolidating nearby *nostalgic cores* to fewer meta-objects called *super nostalgic cores*, *DenForest* can make the $\mathcal{MST}s$ smaller and reduce the overhead of updating clusters.

Definition 17 (Super nostalgic core, SC). *The super nostalgic core is a connected*

component of nostalgic cores in the same stride that become $non\text{-}cores$ together when the window slides by a single stride.

Definition 18 (**ϵ-Neighbors of a super nostalgic core**). *Two super nostalgic cores sc_1 and sc_2 are said to be ϵ-neighbors if there are a pair of points $p \in sc_1$ and $q \in sc_2$ such that $distance(p, q) \leq \epsilon$.*

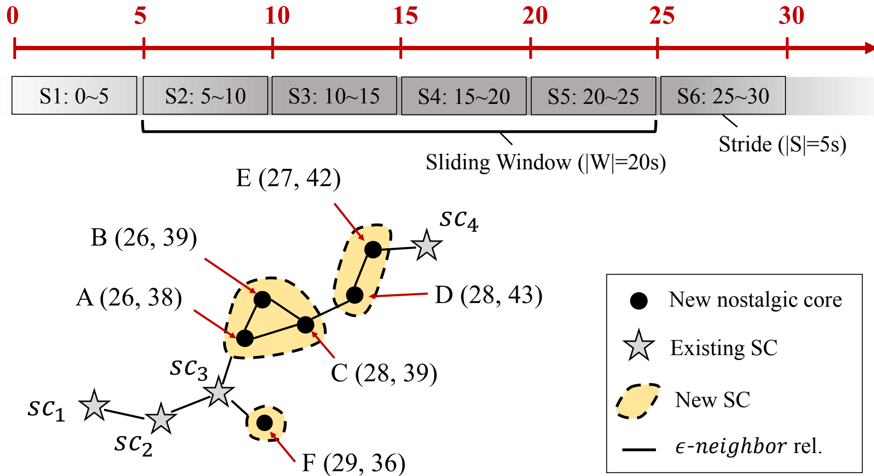


Figure 5.6: Example of Super Nostalgic Cores

In the following example of *super nostalgic cores*, it is assumed that the window and the stride are 20 seconds long and 5 seconds long, respectively, and the sliding window is currently anchored at time 25 covering a time interval (5,25]. This is illustrated in Figure 5.6, where each point is annotated with the timestamp (T) and the core-expiration time (T_c). For example, point $A(26, 38)$ will be ingested at time 26 and will become a $non\text{-}core$ at time 38. *Non-cores* are not shown in the figure.

Example 7. *In Figure 5.6, the four stars ($sc_1 \sim sc_4$) represent super nostalgic cores in the current window. When the window advances by a stride, six new points ($A \sim F$) in the stride $S6$ are added to the window, and they all become nostalgic cores. Among those six points, F is separate from the others by a nostalgic core not in stride S6,*

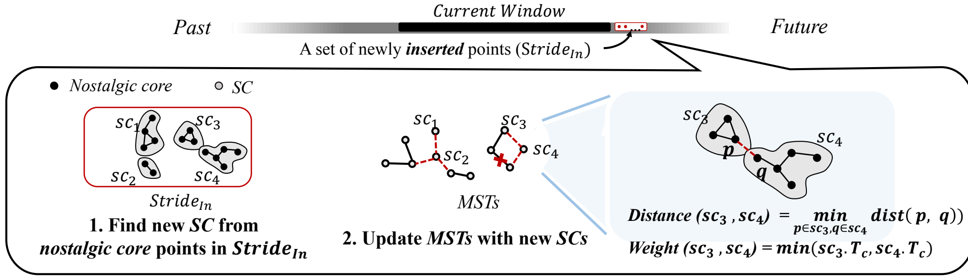


Figure 5.7: Batch-optimized Insert

and it alone forms a super nostalgic core $\{F\}$*. The other points* $A \sim E$ *are in a connected component of* nostalgic cores *in the same stride, but they form two separate super nostalgic cores* $\{A, B, C\}$ *and* $\{D, E\}$*. This is because points A, B, and C will become non-cores at time 40, while points D and E will become non-cores at time 45. The super nostalgic cores* $\{A, B, C\}$ *and* $\{D, E\}$ *are said to be* ϵ*-neighbors because C and D are within the* ϵ*-distance from each other.*

The **batch-optimized** `Insert` algorithm replaces a group of connected *nostalgic cores* with a *super nostalgic core* so that *DenForest* can update clusters more efficiently without compromising the clustering result. The detailed procedure is given below. (See Figure 5.7 for illustration.)

STEP 1 (Finding SCs) When the window slides by a stride and new data points are added, new *nostalgic cores* are found from the data points, and new *super nostalgic cores* are formed from the *nostalgic cores*. The *core-expiration time* of a *super nostalgic core* is set to the time interval of a stride that covers all the *core-expiration times* of its *nostalgic cores*. For example, T_c of the *super nostalgic core* $\{A, B, C\}$ is set to the time interval (35,40]. Besides, each *nostalgic core* maintains a pointer to the adjacent *nostalgic core* with the largest T_c for the batch-optimized deletion. For example, point C maintains a pointer to D, which has the largest T_c among $N_\epsilon(C)$.

STEP 2 (Updating $\mathcal{MST}s$ ***with SCs)*** Each *super nostalgic core* is collapsed to a single vertex in the $\mathcal{MST}s$. A new edge is introduced to each pair of the ϵ-neighbors of

super nostalgic cores, and the weight of the new edge is set to the smaller of their *core-expiration times*. For example, the weight of the edge between two *super nostalgic cores* $\{A, B, C\}$ and $\{D, E\}$ is set to the time interval (35, 40]. If a cycle is formed, then an edge with the smallest weight is removed from the cycle. $Borders$ are updated the same way as the `Insert` algorithm.

The **batch-optimized** `Delete` algorithm works similarly. When the window slides by a stride and old data points are removed, some of the *super nostalgic cores* may become $non\text{-}cores$. For example, when the window slides from a time interval (15, 35] to a time interval (20, 40], *super nostalgic cores* $\{A, B, C\}$ and $\{F\}$ become $non\text{-}cores$. The expired *super nostalgic cores* are removed from the $\mathcal{MST}s$. For each point p that has become a *non-core*, the adjacent *nostalgic core* q with the largest T_c is examined, which is found by following the pointer established in the batch insertion algorithm. If q is still a *nostalgic core*, then p becomes a $border$ point. Otherwise, it becomes a $noise$.

5.3 Clustering Quality of *DenForest*

Two aspects should be considered in evaluating the effectiveness of a clustering method for streaming data over the sliding window. The first is the capability to produce *high-quality* clusters from the current window, and the second is the capability to sustain the quality *efficiently* while the window moves forward. This section evaluates the first aspect of *DenForest*. The second aspect of *DenForest* will be evaluated in Section 6.5.

5.3.1 Clustering Quality for Static Data

A variety of synthetically generated labeled datasets were used for the evaluation of clustering quality. For each dataset, it was assumed that the entire set of data points were contained in the current window from which density-based clusters were produced by *DenForest* as well as DBSCAN for comparison. *DenForest* produces clusters

Table 5.4: Clustering quality on various datasets

Dataset	*DenForest* vs. Label			DBSCAN vs. Label			*DenForest* vs. DBSCAN		
	ARI	AMI	NMI	ARI	AMI	NMI	ARI	AMI	NMI
$Spiral$ [15]	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$R15$ [81]	0.98	0.98	0.98	0.99	0.99	0.99	0.98	0.98	0.98
$Aggr.$ [36]	0.97	0.96	0.97	0.99	0.99	0.99	0.97	0.96	0.97
$Comp.$ [90]	0.94	0.87	0.90	0.94	0.85	0.91	0.98	0.88	0.94
$G2$-2-30 [62]	0.95	0.88	0.90	0.96	0.93	0.93	0.96	0.92	0.94
$G2$-4-30 [62]	0.99	0.97	0.99	1.00	1.00	1.00	0.99	0.97	0.99
$G2$-8-30 [62]	0.99	0.99	0.99	1.00	1.00	1.00	0.99	0.99	0.99
Average	**0.97**	**0.95**	**0.96**	**0.98**	**0.96**	**0.97**	**0.98**	**0.95**	**0.97**

of *nostalgic cores*, while DBSCAN produces clusters of *d-cores* (*i.e.*, cores in DBSCAN's own definition). Although they define *cores* in their own ways, both *DenForest* and DBSCAN define *clusters* the same way.

Three metrics called Adjusted Rand Index (ARI) [45], Adjusted Mutual Information (AMI) [83], and Normalized Mutual Information (NMI) [56] were adopted to measure the clustering quality quantitatively. These metrics have been used widely in various studies to compute the similarity between two cluster memberships (or partitions) [52, 14]. The ARI values range from -1 to 1, with 1 indicating two identical clustering results and -1 indicating no similarity between them. The AMI and NMI are similar to ARI, but their values range from 0 to 1. For each clustering method, the best achievable quality was attempted to be obtained by tuning the density (τ) and the distance (ϵ) thresholds. The clustering results of *DenForest* may vary depending on the ingestion order of data points owing to the way *nostalgic cores* are defined. Thus, for each metric, *DenForest* was ran one hundred times for each dataset each with a random ingestion order and took the average.

Table 5.4 summaries the clustering quality of *DenForest* and DBSCAN tested on seven labeled datasets, which are listed in the first column of the table. In the first and

second groups of three columns are the quality measurements computed with respect to the given labels (*i.e.*, ground truth), which measure the ability of *DenForest* and DBSCAN to produce accurate clustering results. In the third group of three columns are the quality measurements computed with respect to the clustering results from DBSCAN, which measures the ability of *DenForest* to produce the same clustering results as those of DBSCAN. On average, *DenForest* achieved 0.97 (ARI), 0.95 (AMI), and 0.96 (NMI) clustering quality with respect to the given true labels, and achieved 0.98 (ARI), 0.95 (AMI) and 0.97 (NMI) clustering quality with respect to the clustering results from DBSCAN. This demonstrates that considering only the pre-existing data points in the current window does not overly compromise the quality of clusters and helps expedite the clustering process significantly, which will be shown in Section 6.5.

5.3.2 Discussion

The following observation explains this phenomenon: the *nostalgic cores* spatially cover the clusters of the *d-core*. In the density-based clusters, it is assumed that each core point represents a ball with a radius, ϵ, which covers the area within the ϵ-distance from the point. In the dense area (clusters), many balls are piled densely within a small region. The shape of the clusters can then be represented by using only a subset of the points, indicating that the subset of points can spatially cover the other points of the clusters. Thus, the *nostalgic cores* can be explained as a subset of *d-cores*, that are positioned relatively on a front end (recent side) of the window, but spatially cover the *d-cores* producing the indistinguishable clustering results. The *d-cores* may not be spatially well covered with *nostalgic cores* for a $sparse$ region. However, remember that our task is finding the $dense$ area (clusters).

5.3.3 Replaceability

The number of *nostalgic cores* within the distance threshold is critical to the quality of clustering result. This section provides further analysis on the relationship among

the density of a region, the number of *nostalgic cores*, and the clustering quality. First, it will be shown that the number of *nostalgic cores* in a region is linearly correlated with the density of the region (Section 5.3.3.1). Then, it will be shown that *DenForest* and DBSCAN would produce similar clustering results if the region was dense enough (Section 5.3.3.2).

5.3.3.1 Nostalgic Cores and Density

Imagine a set of points scattered in the space and time. The number of points in a space V and a time-interval $(t_1, t_2]$ can be calculated by the equation below with continuous density assumed for simplicity.

$$\int_{t_1}^{t_2} \int_{V} D(\vec{x}, t) \mathrm{d}V \mathrm{d}t \tag{5.4}$$

where $D(\vec{x}, t)$ denotes the density function of space ($\vec{x}$) and time (t). Below a *locally stable* subspace is defined whose number of *nostalgic cores* can be determined with respect to the density of the subspace.

Definition 19 (Locally stable). *A subspace is said to be locally stable if the following equation is satisfied for any point p in the subspace.*

$$\int_{B_\epsilon(p)} D(\vec{x}, t) \mathrm{d}V = Vol(B_\epsilon) \cdot D(p, t)$$

B_ϵ and $B_\epsilon(p)$denote a ball of radius ϵ and a ball of radius ϵ centered at p, respectively. $Vol(B_\epsilon)$ denotes the volume of B_ϵ.

Lemma 8. *In a locally stable subspace, the number of nostalgic cores in any B_ϵ is $D_\epsilon - \tau$, where D_ϵ denotes the number of points in B_ϵ.*

Proof. Let the time interval of the window be $(t_0, t_W]$. By Equation (5.4), D_ϵ is equal to $\int_{t_0}^{t_W} \int_{B_\epsilon} D(\vec{x}, t) \mathrm{d}V \mathrm{d}t$. For a position $\vec{y}$ in the *locally stable* space, define a function $T(\vec{y})$ such that $\tau = \int_{t_0}^{T(\vec{y})} \int_{B_\epsilon(\vec{y})} D(\vec{x}, t) \mathrm{d}V \mathrm{d}t$. $T(\vec{y})$ is a time threshold for *nostalgic core* classification. Among those at the same location as $\vec{y}$, the points inserted after the $T(\vec{y})$ time are classified as *nostalgic cores*, while the points inserted

before that time are not. The number of *nostalgic cores* in B_ϵ can then be defined as $\int_{B_\epsilon}\int_{T(\vec{x})}^{t_W} D(\vec{x},t)\mathrm{d}t\mathrm{d}V$. Therefore, the lemma is proved as follows.

$$\begin{aligned}
&\int_{B_\epsilon}\int_{T(\vec{x})}^{t_W} D(\vec{x},t)\mathrm{d}t\mathrm{d}V = D_\epsilon - \int_{B_\epsilon}\int_{t_0}^{T(\vec{x})} D(\vec{x},t)\mathrm{d}t\mathrm{d}V \\
=&D_\epsilon - \int_{B_\epsilon} \tau/Vol(B_\epsilon)\mathrm{d}V \qquad (\textit{by the locally stable condition}) \\
=&D_\epsilon - \tau.
\end{aligned}$$

□

This lemma indicates that in the region with a sufficiently high density ($D_\epsilon \gg \tau$), there will be many *nostalgic cores* in any B_ϵ.

5.3.3.2 Nostalgic Cores and Quality

The *d-cores* of DBSCAN play two important roles : (1) spatially $covering$ the clustered region and (2) $connecting$ the neighboring points. If *nostalgic cores* completely *replace d-cores* playing these roles, *DenForest* and DBSCAN will produce an identical result. This replaceability is correlated with the number of *nostalgic cores* in B_ϵ.

For a *d-core* point p, let $NC_\epsilon(p)$ be a set of *nostalgic cores* in $B_\epsilon(p)$. Recall that *nostalgic cores* are a subset of *d-cores*.

Definition 20 (Completely replaceable). *A d-core p is said to be completely replaceable by $NC_\epsilon(p)$, if the following conditions are satisfied.*

$$B_\epsilon(p) \subseteq \bigcup_{q \in NC_\epsilon(p)} B_\epsilon(q) \qquad (Coverage)$$

A DenGraph's subgraph whose vertex set is $NC_\epsilon(p)$ is connected. $\quad (Connectivity)$

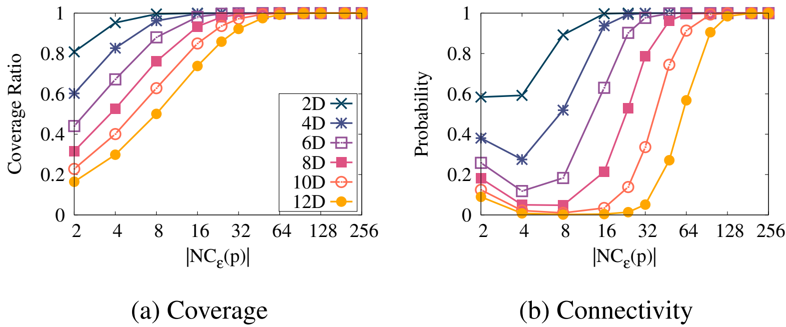
Theorem 4. *For any d-core p, if it can be completely replaced by $NC_\epsilon(p)$, then both DenForest and DBSCAN produce an identical clustering result.*

Proof. The $Coverage$ condition guarantees that the area covered by *nostalgic cores* contains all the borders and *d-cores* of DBSCAN. The $Connectivity$ condition guarantees that all the *d-cores* in a cluster of DBSCAN are included in a cluster of *DenForest*. □

This theorem clearly states that the two conditions of Definition 20 are relevant to clustering quality. Now, let us find out how they are correlated with the cardinality of $NC_\epsilon(p)$. Figure 5.8 shows the coverage ratio of a point p and the probability of the subgraph composed of $NC_\epsilon(p)$ being connected, with respect to the $|NC_\epsilon(p)|$ and the dimensionality of space. The *Monte Carlo* method [63] was adopted, and $NC_\epsilon(p)$ is populated uniformly around p. The coverage ratio is calculated by the following equation.

$$Coverage\ Ratio = \frac{Vol(B_\epsilon(p) \cap (\bigcup_{q \in NC_\epsilon(p)} B_\epsilon(q)))}{Vol(B_\epsilon)}$$

The general trend is that, as $|NC_\epsilon(p)|$ increases, both the coverage and the connectivity increase. For example, in a 2D space, if $D_\epsilon - \tau \geq 16$, then the clustering result of *DenForest* will be nearly identical to that of DBSCAN. This is because 16 or more *nostalgic cores* around a *d-core* p can replace it completely.



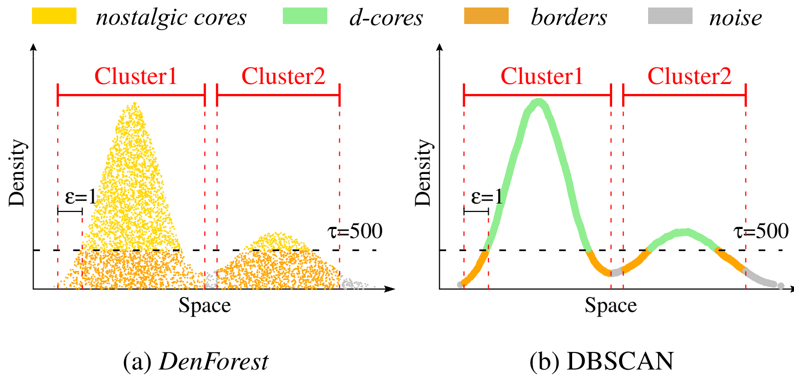
(a) Coverage (b) Connectivity

Figure 5.8: Coverage and Connectivity w.r.t. $|NC_\epsilon|$

In *interior regions* of a cluster where the density is sufficiently high ($D_\epsilon \gg \tau$), there would be many *nostalgic cores* within the ϵ-distance. Thus, the *nostalgic cores* would replace *d-cores* well with the high coverage ratio and the high probability of being connected. In *boundary regions* of a cluster where the density is not so high ($D_\epsilon \approx \tau$), there might not be enough points in NC_ϵ, and the *nostalgic cores* would not replace *d-cores* so well. However, the boundary regions are fundamentally unstable, and they seldom affect the quality of clustering result.

5.3.4 1D Example

Here is an example for the better understanding about the internals of the cluster in *DenForest*. Consider that two clusters emerge in an one-dimensional space where each point is randomly generated over time from a bi-modal distribution, $\{3\mathcal{N}(0,1) + 2\mathcal{N}(6,1.5)\}/5$. The window is filled with 5 K data points preserving the time order with $|stride|$=1.



(a) *DenForest* (b) DBSCAN

Figure 5.9: Density-Based Clusters produced by *DenForest* and DBSCAN

DenForest and DBSCAN are then run to detect two clusters by setting ϵ=1 and τ=500 using the data points in the window. Figure 5.9 presents the description of the detected clusters. The y-axis denotes the density of each data point. For each point in the window, *DenForest* calculates the density as the number of previously inserted points within the ϵ-distance, while DBSCAN calculates the density including the num-

ber of points inserted afterwards as well.

In Figure 5.9a, the data points of *DenForest* are distributed like a plane because a group of data points in very close proximity tends to have widespread density values according to their ingestion time. On the other hand, the data points of DBSCAN are distributed like a line in Figure 5.9b, because a group of data points in very close proximity shows similar densities. It can be noticed that the *nostalgic cores* cover the space where the *d-cores* exist. Furthermore, a set of the *d-cores* not in the *nostalgic cores* ($\{$*d-cores* $\}\setminus\{$*nostalgic cores* $\}$) is included into the clusters as $borders$ in *DenForest* (that are spatially covered by *nostalgic cores*). Therefore, *DenForest* and DBSCAN produce the nearly identical cluster memberships.

Chapter 6

Evaluation

This chapter analyzes the performance of $\mathcal{DISC}$ and *DenForest* by comparing them with the existing incremental density-based clustering methods.

6.1 Real-World Datasets

In the experiments, the following five real-world datasets were used to evaluate the proposed methods.

DTG is a dataset collected from digital tachograph devices attached to commercial vehicles in a metropolitan city [25]. A record was generated from each vehicle every 10 seconds, and each record included the time, location, speed, and acceleration of the vehicle. The 2D coordinates (p_{lat}, p_{lon}) were used in the experiments, where p_{lat} and p_{lon} are the latitude and the longitude fields, respectively. The total number of records is approximately 300 million.

GeoLife is a GPS trajectory dataset collected from 182 users over a period of four years [92]. Each record includes the time and the location of each user. The 3D normalized coordinates $(p_{lat}, p_{lon}, p_{alt}/300,000)$ were used in the experiments, where p_{alt} is the altitude field. The total number of records is approximately 24.8 million.

COVID-19 consists of geo-tagged tweets about the novel *coronavirus* from March

to September 2020 [57]. Each record includes the time and location of a tweet around the world. The 2D coordinates (p_{lat}, p_{lon}) were used to denote the spatial location. The total number of records is 210 thousand.

IRIS is a dataset of earthquake events that occurred around the world from 1960 to 2019 [46]. The 4D normalized coordinates $(p_{lat}, p_{lon}, p_{dep}/10, p_{mag} \times 10)$ were used in the experiments, where p_{dep} and p_{mag} are the depth and the magnitude fields, respectively. The total number of records is approximately 1.8 million.

Household is a dataset of the electric energy consumption in a household over a period of four years [26]. Each record includes seven fields related to the power and voltage information. The 7D coordinates normalized by the variance of the fields were used in the experiment. The total number of records is approximately 2 million.

6.2 Competing Methods

6.2.1 Exact Methods

The proposed methods in this dissertation are compared with two incremental methods that can produce exactly the same clustering result as that of DBSCAN : Incremental DBSCAN, and Extra-N. Incremental DBSCAN (or IncDBSCAN in short) is an incremental version of DBSCAN that supports insertion and deletion of an individual data point [27]. Its version optimized with MS-BFS was used in the experiment. Extra-N [87] is another clustering method that supports incremental updates under the sliding window model.

6.2.2 Non-Exact Methods

$\mathcal{DISC}$ and *DenForest* are compared with non-exact methods that produce the approximate or summarized clustering results for DBSCAN : ρ-double-approximate DBSCAN, DBSTREAM, SDStream, and StreamSW. ρ-double-approximate DBSCAN [34] produces an approximate clustering result by managing grids. DBSTREAM [41], SD-

Stream [68], and StreamSW [74] are summarization-based methods that produce summarized clustering results using statistical information. Besides, EDMSTREAM [37] is also compared, which is a clustering algorithm for data streams based on the Density-Peak algorithm [69].

6.3 Experimental Settings

All the experiments were conducted on a stand-alone machine with a Ryzen 7 1700 8-Core Processor, 64 GB RAM, and a 256 GB solid-state drive, running Ubuntu 18.04 LTS. Since each dataset was preloaded into the memory, the disk did not affect the performance during the experiments. The elapsed times were measured using the `System.nanoTime` function. Each of all the measurements presented in this section is the average of five runs. Except for EDMSTREAM[1], all the aforementioned clustering methods as well as an in-memory version of the R-tree index were implemented in Java with JDK 1.8.0-121.

Throughout the experimental evaluation, the *count-based* sliding window model was adopted where its parameters, *window size* and *stride*, are measured in terms of the number of data points rather than time duration. This model enables us to control the amount of workload and calibrate the experimental settings with more ease. The ingestion order of data points still follows strictly the time stamp of the data records.

6.4 Evaluation of $\mathcal{DISC}$

In Sections 6.4.2 and 6.4.3, the performance characteristics of $\mathcal{DISC}$ are analyzed in comparison with existing density-based clustering methods, DBSCAN [28], IncDBSCAN [27], and EXTRA-N [87]. In Section 6.4.4, $\mathcal{DISC}$ is compared with non-exact methods : DBSTREAM [41], EDMSTREAM [37] and ρ-double-approximate DBSCAN [34].

[1]The Java code is available in https://github.com/ShufengGong/EDMStream.

Dataset	density (τ)	distance (ϵ)	$\lvert window \rvert$
DTG	372	0.002	2M (~10 min)
$GeoLife$	7	0.01	200K (~fortnight)
$COVID$-19	5	1.2	15K (~fortnight)
$IRIS$	9	2	200K (~decade)

Table 6.1: Threshold values and window sizes

6.4.1 Parameters

Table 6.1 summarizes the threshold values and the default window sizes chosen for each dataset. For the DTG dataset, the ground traffic monitoring example was adopted to set the distance (ϵ) and density (τ) thresholds. The distance threshold was set to be small enough to distinguish roads in close proximity, and the density threshold was set to the average number of points within the distance threshold. For the other datasets, the parameter settings used by the previous work based on a K-distance graph [28, 71] were adopted. The sliding window sizes were set to a fraction of each dataset, roughly corresponding to a chosen time duration.

6.4.2 Baseline Evaluation

This section compares the overall performance of $\mathcal{DISC}$ with such exact clustering methods as DBSCAN, IncDBSCAN, and EXTRA-N with respect to elapsed time. Since DBSCAN is a clustering algorithm designed for a static database, it was used as the baseline method rather than a target of direct comparison in the experiments, and measured the performance of the other methods in relation to that of DBSCAN.

Figures 6.1 and 6.2 show the relative speedup of $\mathcal{DISC}$, IncDBSCAN and EXTRA-N over DBSCAN for the four real-world datasets, with a varying size of stride and window, respectively. (As for the absolute performance measurements, the average elapsed times taken by DBSCAN were 102s, 523s, 496ms, and 533s for the four datasets, re-

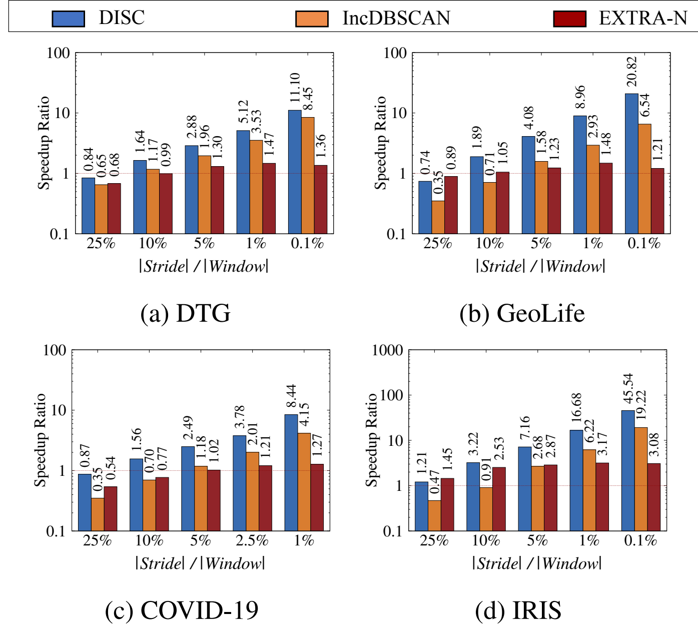


Figure 6.1: Relative speedup over DBSCAN with a varying size of stride

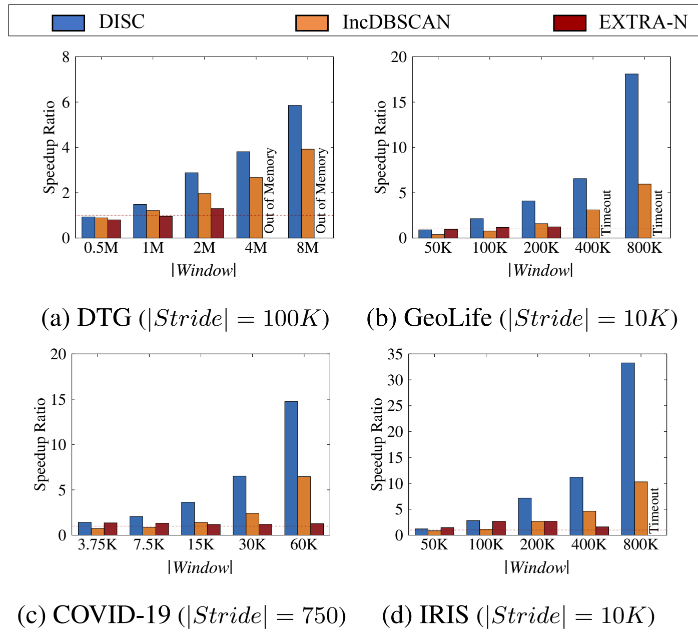


Figure 6.2: Relative speedup over DBSCAN with a varying size of window

spectively, in Figure 6.1.) For each of the four clustering methods including DBSCAN, the average elapsed time taken to update clusters was measured when the sliding window advanced by a single stride.

The execution time of DBSCAN remained unaffected by a varying ratio of stride to window, because it recomputed clusters from scratch whenever the sliding window advanced. In contrast, the execution time of the other three methods was affected significantly by the ratio. In particular, IncDBSCAN and $\mathcal{DISC}$ updated clusters incrementally focusing on the data points leaving and entering the sliding window. Consequently, their execution time tended to decrease as the stride shrank smaller.

In the case of EXTRA-N, however, its speedup over DBSCAN started being saturated earlier as the stride shrank smaller or the window grew larger. As is shown in Figure 6.2, when the sliding window was large, EXTRA-N exceeded the memory capacity or was terminated forcefully after ten hour execution. This is because EXTRA-N maintains as many sub-windows as the number of strides fitting in a single window so that it can keep track of the local neighbors of individual data points. Thus, when the ratio of stride to window was too high, it suffered from the steep increase of memory consumption, and the cost of maintaining too many sub-windows outweighed the benefit from avoiding range searches.

When the stride was no larger than 10 percent of the sliding window, $\mathcal{DISC}$ was the best performer among all the clustering methods compared including DBSCAN. In Figure 6.1, for example, the speedup of $\mathcal{DISC}$ over the second-best performer ranged from 27% (in IRIS with a 10% stride) to 318% (in GeoLife with a 0.1% stride). There was no clear second-best performer in this range of stride sizes for the datasets.

The most noteworthy feature of $\mathcal{DISC}$ observed from this set of experiments was that its benefit is amplified particularly when applied to *finer-grained* incremental clustering with the sliding window advancing frequently in a small stride. On the other hand, when the stride was as large as 25 percent of the sliding window, all the three incremental methods performed poorly. Their execution times were comparable to that

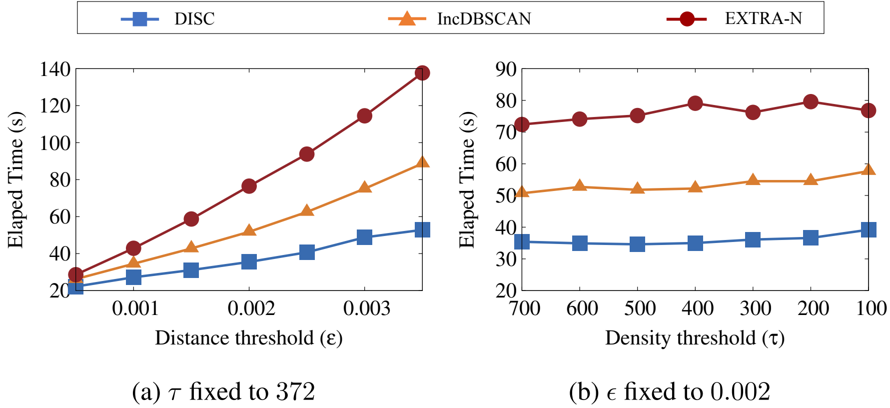
of DBSCAN at best or even worse than that. Given that an efficient method is desired more for finer-grained incremental clustering, it clearly attests that $\mathcal{DISC}$ is an effective tool for the task of clustering fast evolving streaming data.

6.4.3 Drilled-Down Evaluation

Having presented the baseline evaluation of $\mathcal{DISC}$, further analyses to understand its performance characteristics in more detail under various parameter settings and the effects of the proposed optimization techniques were provided. Unless stated otherwise, the experiments were carried out under the same default settings as shown in Table 6.1.

6.4.3.1 Effects of Threshold Values

The density-based clustering is governed by two thresholds, density (τ) and distance (ϵ), as they determine which points are cores. Therefore, these parameters inherently have a critical influence over not only the quality of clustering results but also the cost of cluster discovery.



(a) τ fixed to 372

(b) ϵ fixed to 0.002

Figure 6.3: Threshold effects : distance (ϵ) and density (τ)

Figures 6.3(a) and 6.3(b) show the elapsed times taken by the three incremental clustering methods for the DTG dataset with a varying distance threshold (ϵ) and with a varying density threshold (τ), respectively. The stride size was fixed to 5% of the

window size. The elapsed times of all three methods were elongated as the value of ϵ increased or the value of τ decreased. This is because a longer distance threshold allowed data points to have more ϵ-$neighbors$ and a lower density threshold produced a larger population of core points. However, the impact of τ on the elapsed time was not as significant as anticipated. Note that the performance of $\mathcal{DISC}$ was much more stable and efficient than the others over the entire spectrum of ϵ values and τ values tested. The same trend was observed from the other datasets as well.

6.4.3.2 Insertions vs. Deletions

In an attempt to analyze separately the effect of insertions and deletions on the clustering performance, hypothetical scenarios was created where one stride advance of the sliding window only accepted new data points to the window or only expelled existing points from the window. Figure 6.4 compares the elapsed times taken by $\mathcal{DISC}$ and IncDBSCAN, when 5% of data points were inserted to or deleted from the current window in each case of the four datasets. EXTRA-N was not included in this experiment because its sub-window based approach does not distinguish insertions from deletions. Though IncDBSCAN and $\mathcal{DISC}$ both spent more time on processing

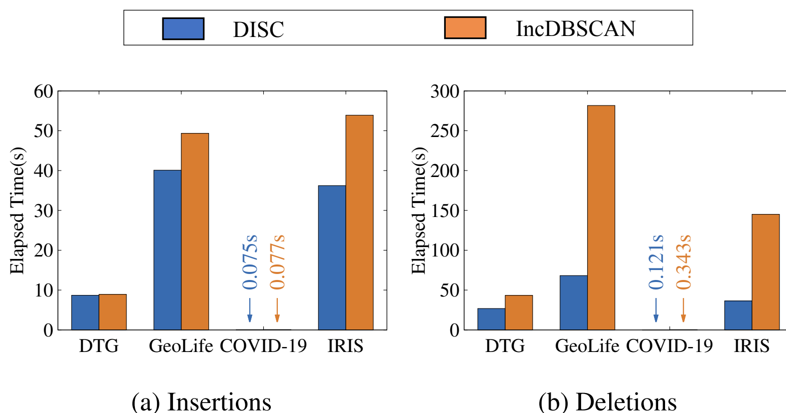


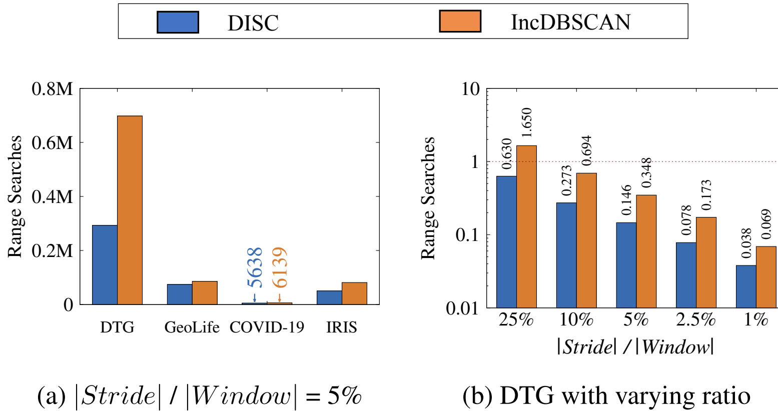
Figure 6.4: Insertions vs. Deletions

the deletion-only workloads than the insertion-only workloads, $\mathcal{DISC}$ handled dele-

tions much more gracefully. While IncDBSCAN was 314% slower on average for the deletion-only workloads than the insertion-only workloads, $\mathcal{DISC}$ was just about 120% slower. Overall, $\mathcal{DISC}$ achieved speedups over IncDBSCAN for both insertions and deletions across all the datasets. The factor of improvement was in the range of 1.03 (for insertions of DTG) and 4.30 (for deletions of GEO).

6.4.3.3 Range Searches

The number of range searches executed by IncDBSCAN and $\mathcal{DISC}$ was counted in order to understand how much the clustering methods were affected by the costly search operations. Unlike DBSCAN, as a static approach, that always invokes as many range searches as the number of data points in the current sliding window, the number of range searches required by IncDBSCAN and $\mathcal{DISC}$ is dependent on the stride sizes.



(a) $|Stride|$ / $|Window|$ = 5% (b) DTG with varying ratio

Figure 6.5: Range searches executed

Figure 6.5(a) shows the number of range searches carried out by $\mathcal{DISC}$ and IncDBSCAN with the ratio of stride to window fixed to 5%. $\mathcal{DISC}$ invoked a smaller number of range searches than IncDBSCAN across all the four datasets. Figure 6.5(b) compares the two methods relatively in comparison with DBSCAN for the DTG dataset. $\mathcal{DISC}$ was superior to IncDBSCAN as well as DBSCAN in the number of range search invocations consistently across all the stride-to-window ratios tested. Figure 6.5(b)

coupled with Figure 6.1(a) clearly indicates that the number of range searches has a direct impact on the performance of DBSCAN, IncDBSCAN, and $\mathcal{DISC}$.

6.4.3.4 MS-BFS and Epoch-Based Probing

The MS-BFS and epoch-based probing presented in Section 4.2 are optimization techniques proposed for $\mathcal{DISC}$ so that the cost of checking the density-connectedness of minimal bonding cores can be further reduced. These two techniques can be applied independently of each other, and so can their effect be evaluated separately. Figure 6.6 shows the elapsed times taken by $\mathcal{DISC}$ for each dataset, when neither optimization was applied, only the epoch-based probing was applied, only the MS-BFS was applied, and both were applied. The elapsed times were measured with the stride size fixed to 5% of the window size.

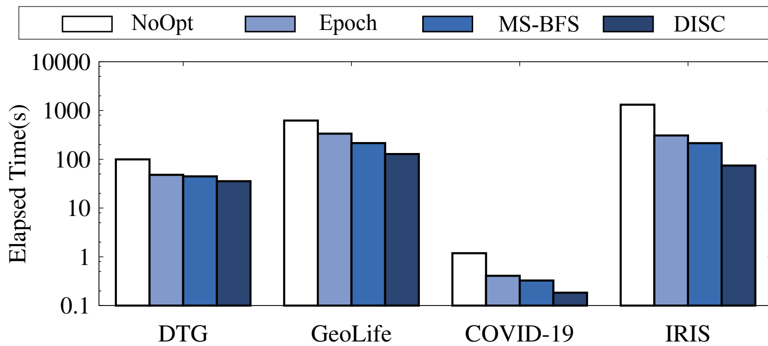


Figure 6.6: Effects of optimizations

Each of the two optimization techniques achieved substantial reduction in the elapsed times even when they are applied alone separately. Between the two, the MS-BFS was slightly more effective than the epoch-based probing consistently over all the datasets. Apparently, the best performance was attained when both the optimization techniques were applied together, yielding more than an order of magnitude speedup in the case of the IRIS dataset.

6.4.4 Comparison with Summarization/Approximation-Based Methods

DBSTREAM [41] and EDMSTREAM [37] are the state of the art summarization-based methods known for the low latency and high quality of clustering. The ρ-double-approximate DBSCAN (or ρ_2-DBSCAN in short) is another approximate clustering method, which is the dynamic version of ρ-approximate DBSCAN [33, 34]. $\mathcal{DISC}$ was compared with these three clustering methods to evaluate the trade-off between the processing speed and the quality of clustering. Their capability of capturing the detailed shape of clusters was focused on by adopting rather small values for the distance threshold ϵ.

Two datasets, DTG and Maze, were used for this evaluation. For the real dataset DTG, the clustering results from DBSCAN were used as the true labels. The synthetic dataset Maze was created by placing 100 random seeds in the 2-dimensional space. They spread out over time such that the trajectory of each seed was mapped to a single cluster. When the window size increased, trajectories became longer and closer to one another, and consequently the shape of clusters grew more complicated. Each point in the Maze dataset was manually labeled so that each trajectory could be identified clearly as a separate cluster.

To evaluate the quality of clustering results, the Adjusted Rand Index (ARI) [45] was measured with a varying size of sliding window. The ARI measures how close the clustering results from different methods are to the true labels, and the measurements are in the range of -1 (lowest) to 1 (highest).

Figure 6.7 shows the quality measurements and the per-point update latency observed in the Maze dataset. The stride was 5% of the window size. (Since no deletion was supported by the summarization-based methods, only the insertion latency was measured for DBSTREAM and EDMSTREAM.) DBSTREAM and EDMSTREAM were evaluated with parameter settings that helped them achieve the best ARI. The same thresholds, τ and ϵ, were used for both ρ_2-DBSCAN and $\mathcal{DISC}$. The approximation parameter (ρ) of ρ_2-DBSCAN was set to 0.1 and 0.001 for low and high accuracy,

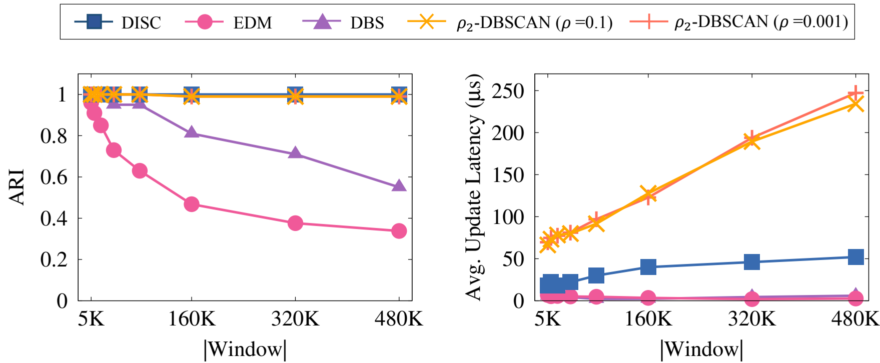


Figure 6.7: Maze: ARI and Update Latency

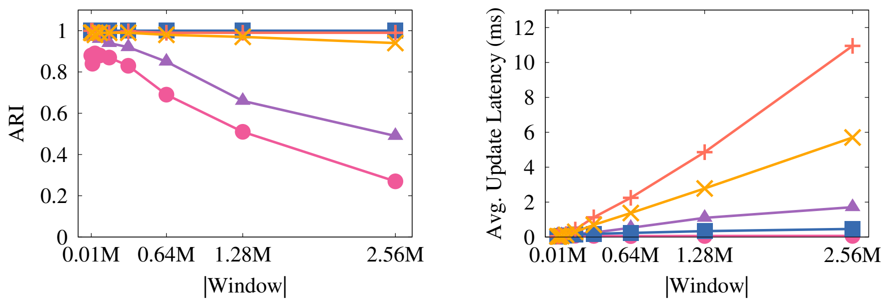


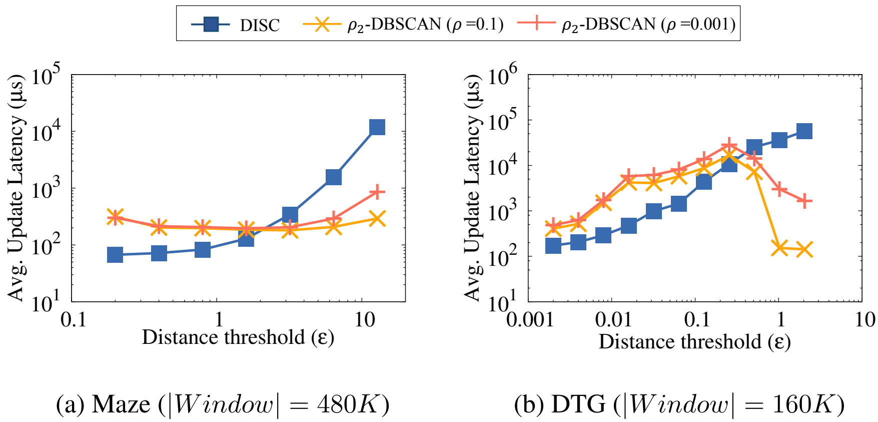
Figure 6.8: DTG: ARI and Update Latency

respectively.

The summarization-based methods, EDMSTREAM and DBSTREAM, were much faster than the others but their clustering quality (measured in ARI) deteriorated very quickly as the sliding window grew larger. To achieve high ARI for a large window, summarization-based methods need to connect micro-clusters correctly.[2] EDMSTREAM connected them well when it dealt with a small number of large micro-clusters, but it did not do so well for a large number of small micro-clusters. DBSTREAM achieved better ARI than EDMSTREAM by utilizing additional information about the connectivity among micro-clusters, although that was not enough to sustain its ARI level. Both ρ_2-DBSCAN and $\mathcal{DISC}$ were able to detect accurate clusters but ρ_2-DBSCAN was up to five times slower than $\mathcal{DISC}$.

[2]Micro-cluster is a summarized representation of a set of adjacent points.

Similar trends were observed in the DTG dataset as shown in Figure 6.8 except for DBSTREAM, which was considerably slower than $\mathcal{DISC}$ across all the window sizes tested. This is because it has to manage a large number of micro-clusters to catch the details of fine-grained clusters. Although ρ_2-DBSCAN yielded high ARI comparable with that of $\mathcal{DISC}$, ρ_2-DBSCAN was much slower than all the other methods. With a larger approximation parameter (ρ), it ran faster but it was still slower than all the other methods.

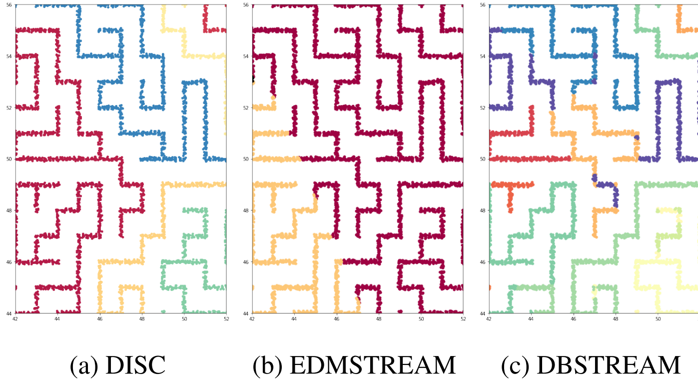


(a) Maze ($|Window| = 480K$) (b) DTG ($|Window| = 160K$)

Figure 6.9: Update Latency with varying ϵ

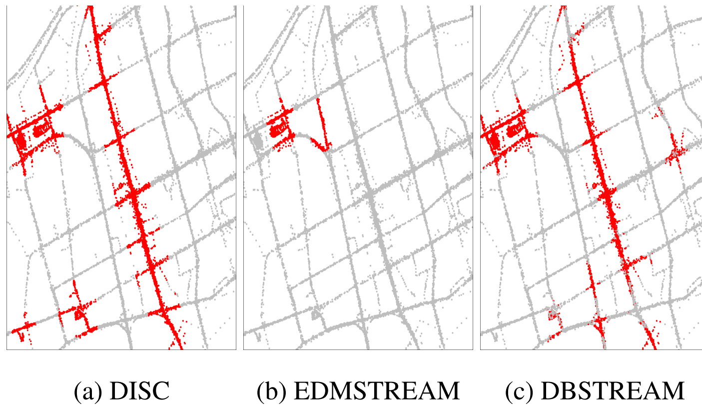
Figure 6.9 compares $\mathcal{DISC}$ and ρ_2-DBSCAN with a varying distance threshold ϵ. the cluster update latency when the stride was 5% of the window size was measured. The overall trends were similar to those reported by Schubert *et al.* [71] about the static version of ρ_2-DBSCAN. For both datasets, $\mathcal{DISC}$ outperformed ρ_2-DBSCAN considerably with smaller ϵ values. $\mathcal{DISC}$ was outperformed by ρ_2-DBSCAN only when $\epsilon \geq 3.2$ for Maze and $\epsilon \geq 0.512$ for DTG. Beyond those crossover points, however, the clustering results were completely meaningless. Those distance thresholds were simply too large and only one huge cluster was detected covering all or almost all the data points in the window.

Figure 6.10 and 6.11 illustrates the clusters discovered by different methods. Since ρ_2-DBSCAN and $\mathcal{DISC}$ produced the same (or almost the same) clusters, the results



(a) DISC (b) EDMSTREAM (c) DBSTREAM

Figure 6.10: Illustration of clusters found in Maze ($|W|$=480K)



(a) DISC (b) EDMSTREAM (c) DBSTREAM

Figure 6.11: Illustration of clusters found in DTG ($|W|$=2.56M)

from ρ_2-DBSCAN is omitted in the figure . Figures 6.10 show clusters (in different colors) found in Maze by $\mathcal{DISC}$, EDMSTREAM, and DBSTREAM, respectively. Only $\mathcal{DISC}$ detected a connected component as a separate cluster correctly. Figures 6.11 show clusters (marked in red color) found in DTG by the three methods. Only $\mathcal{DISC}$ detected the same clusters that matched those found by DBSCAN.

The experiments above confirm that $\mathcal{DISC}$ can detect clusters of high resolution with relatively low cost. Although the summarization-based methods can process streaming data at high speed, the quality of clustering results deteriorates significantly when the sliding window becomes large. The experiments also demonstrate that

ρ_2-DBSCAN, the dynamic version of approximate DBSCAN, consumes an exceedingly high amount of computing time to detect clusters of high resolution, which was confirmed for its static version by the previous work [71]. ρ_2-DBSCAN was outperformed by $\mathcal{DISC}$ significantly for any practically useful range of distance thresholds.

6.5 Evaluation of *DenForest*

In this section, the performance characteristics of *DenForest* are analyzed in comparison with aforementioned competing methods as well as $\mathcal{DISC}$. The baseline evaluation and the drilled-down evaluation for various parameters are presented in sections 6.5.2 and 6.5.3. The *DenForest* method without the batch-optimization (presented in Section 5.2.5) is denoted by *DenForest-NO*. For ρ-double-approximate DBSCAN, two approximation parameters were chosen in the experiments, $\rho = 0.001$ and $\rho = 0.1$, for nearly accurate and less accurate clusters, respectively. The clustering results produced with these two parameters are denoted by $Approx\text{-}High$ ($\rho = 0.001$) and $Approx\text{-}Low$ ($\rho = 0.1$).

6.5.1 Parameters

The density (τ) and the distance (ϵ) thresholds of all the methods were set according to the following scheme. For the DTG and GeoLife datasets, a traffic monitoring example was adopted to set the thresholds. The distance threshold was set to 0.002 degrees (or approximately 222 meters) so as to be small enough to distinguish two close but separate roads. The density threshold was set to the average number of points within the distance threshold to identify congested regions.

For the other datasets, a heuristic scheme was adopted based on the K-distance graph used in the previous studies [28, 71]. The default settings of the density and distance thresholds as well as the window size for each dataset are summarized in Table 6.2.

Table 6.2: Threshold values and window sizes

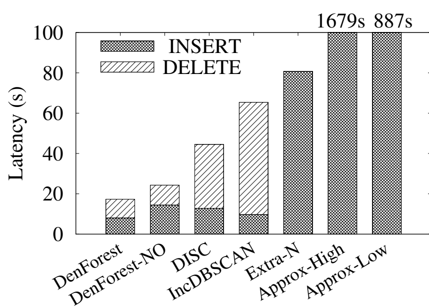
Dataset	dim	density (τ)	distance (ϵ)	$\|window\|$
DTG	2D	372	0.002	2M ($\sim$10 min)
$GeoLife$	3D	765	0.002	0.1M ($\sim$ week)
$IRIS$	4D	8	2	0.2M ($\sim$ decade)
$Household$	7D	14	0.3	0.5M ($\sim$ year)

6.5.2 Baseline Evaluation

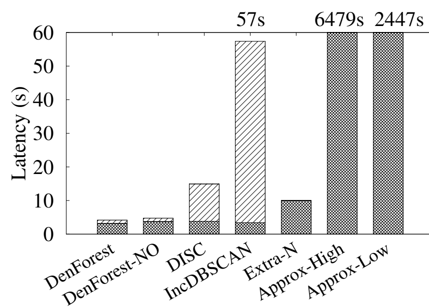
For the baseline performance evaluation, the update latency of each clustering method under the sliding window model is presented in Figure 6.12. For each dataset, the time taken to update clusters was measured when the window advanced by a single stride. The stride size was set to 5% of the window size, whose default settings are given in Table 6.2. The update latency is broken down to the insertion and deletion latency, and each measurement is the average of five runs. Since Extra-N does not support insertion and deletion operations separately, only a combined latency is shown in the figure.

DenForest and its non-optimized version (*DenForest-NO*) outperformed all the other methods. *DenForest* was up to 3.5 times faster than the second-best performer (DISC in the case of Household). IncDBSCAN yielded poor performance particularly for the GeoLife dataset. The reason is the GeoLife dataset is highly skewed in certain areas, which elongates the time taken for range searches significantly. DISC also relies on range searches but it is less affected by the skewedness of the dataset. This is because it takes advantage of optimized range searches such as epoch-based probes that reduce the redundant retrieval of data points.

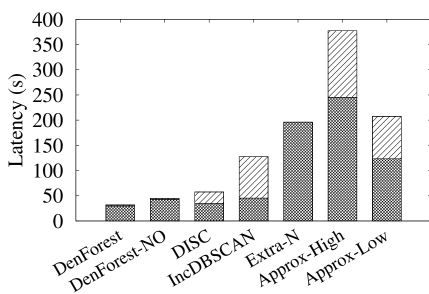
Approx-Low and Approx-High showed poor performance for all the datasets. To determine whether a point is a $core$ or not, the approximate method invokes a number of approximate range counting queries. Not only is it a major bottleneck but also it is aggravated as τ gets larger or as the number of dimensions increases. For the



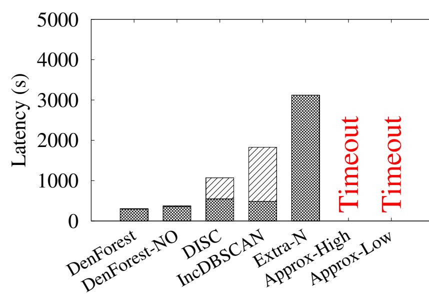
(a) DTG



(b) GeoLife



(c) IRIS



(d) Household

Figure 6.12: Update Latency (|Stride|/|Window|=5%)

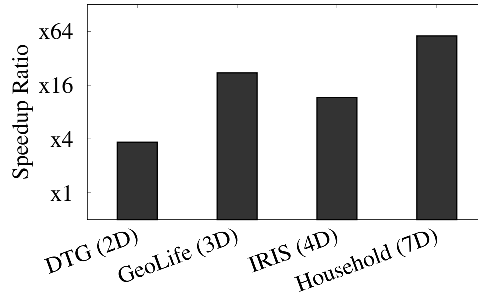
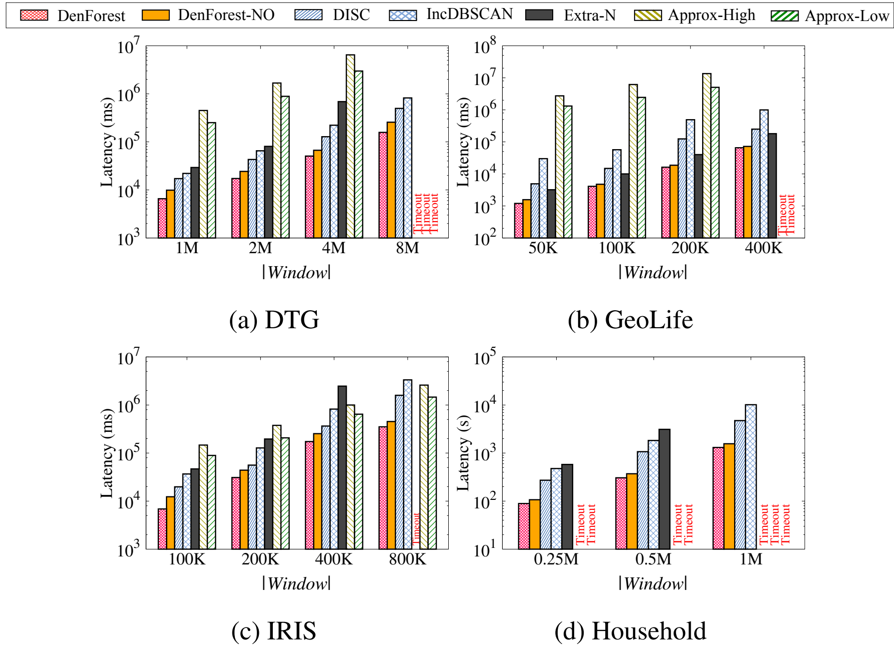


Figure 6.13: Speedup of `Delete`

Household dataset, it did not even terminate within the allotted time of ten hours.

For all the datasets, the deletion latency of *DenForest* was much lower than the other methods. Except for the DTG dataset, the cost of deletion of *DenForest* was almost negligible. Figure 6.13 shows the speedup ratio of the deletion operation by *DenForest* when compared to the second-best performer in the log scale. The deletion time taken for processing a single stride was measured with the stride size set to 5% of the default window size. For the GeoLife dataset, the measurement of the third best performer (DISC) was used because the second best performer (Extra-N) does not support the insert and delete operations separately.

For a $vanishing$ core, *DenForest* simply cuts the links incident to the $vanishing$ core in $\mathcal{MST}$ to update the connectedness of the cluster, while DISC (second best performer) and IncDBSCAN invoke consecutive range searches in a BFS way. This contributes to the major performance improvement by *DenForest* over the other methods. Furthermore, it is less affected by the dimensionality, since the deletion by *DenForest* does not involve any range search. Consequently, the performance gap in the deletion operations increased with the increase of dimensionality. For the 7-dimensional Household dataset, *DenForest* achieved 56 times higher deletion speed than DISC.



(a) DTG (b) GeoLife

(c) IRIS (d) Household

Figure 6.14: Varying size of window (|Stride|/|Window|=5%)

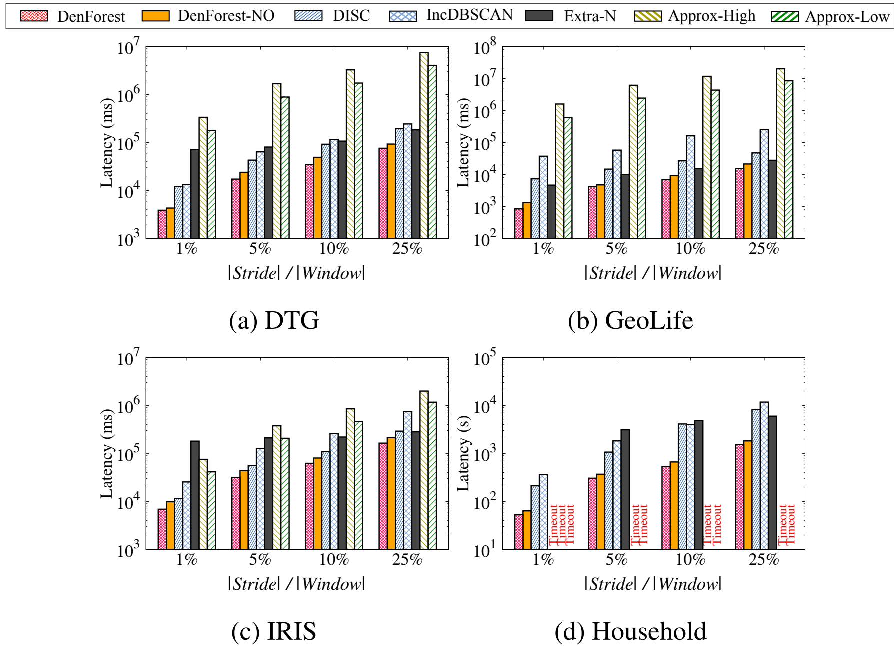
6.5.3 Drilled-Down Evaluation

Further analyses under various parameter settings are provided to understand the performance characteristics of *DenForest* in detail.

6.5.3.1 Varying Size of Window/Stride

The window size and the stride size can vary depending on the applications. Thus, the update latency was measured under various window sizes (Figure 6.14) and various stride sizes (Figure 6.15). The density (τ) and distance (ϵ) thresholds were set to the values in Table 6.2. Both *DenForest* and *DenForest-NO* outperformed the other clustering methods significantly with a wide margin for all the window sizes and for all the stride sizes. For some of the datasets, Extra-N and Approx-Low/High did not terminate within ten hours.

DenForest outperformed *DenForest-NO* (without batch optimization) across the



(a) DTG

(b) GeoLife

(c) IRIS

(d) Household

Figure 6.15: Varying size of stride

entire spectrum of the window sizes and the stride sizes. The batch optimization of *DenForest* effectively lowered the cost of updating clusters by keeping the $\mathcal{MST}s$ smaller. On the other hand, the batch optimization incurs an additional overhead for managing *super nostalgic cores* and their neighbors. The amount of improvement by the batch optimization is also affected by the locality of data points in the same stride. The higher locality results in the more improvement. Therefore, increasing the stride size does not always contribute to performance gain by the batch optimization. On average, the batch optimization improved the performance about 25%.

6.5.3.2 Effect of Density and Distance Thresholds

In this section, the DTG dataset was used to measure the effect of the density and distance thresholds on the performance. The insertion and deletion times taken to process a single stride were measured for each clustering method. The window size was set to

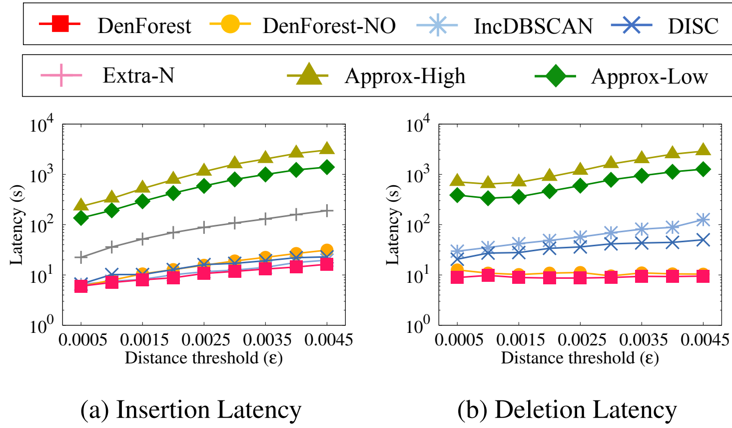


Figure 6.16: Varying ϵ for the DTG dataset

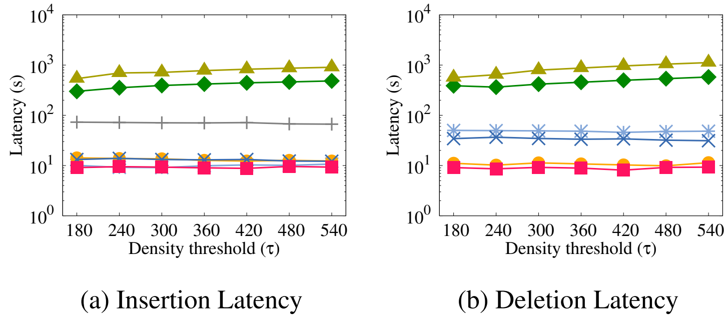


Figure 6.17: Varying τ for the DTG dataset

two million points, and the stride size was set to 5% of the window size.

Figure 6.16 shows the insertion and deletion latency with a varying distance threshold (ϵ). The density threshold was set to the default value in Table 6.2. The larger ϵ value generally requires the more time for range searches. Thus, the insertion and deletion latency increased as the ϵ threshold increased for all the clustering methods except for deletion by *DenForest* and *DenForest-NO*. The reason is of course they do not require any range search for deletion.

A similar experiment was conducted by varying the density threshold (τ) with the distance threshold fixed to the default value. Figure 6.17 shows that the density threshold hardly affected the performance except for Approx-High and Approx-Low,

which slowed down as the density threshold increased. Approx-High (τ=540) was about 80% slower than Approx-High (τ=180), and Approx-Low (τ=540) was about 50% slower than Approx-Low (τ=180). The reason is that the approximate method takes more time to determine whether a point is a core or not as the density threshold increases. A similar trend was also observed in the previous study [84].

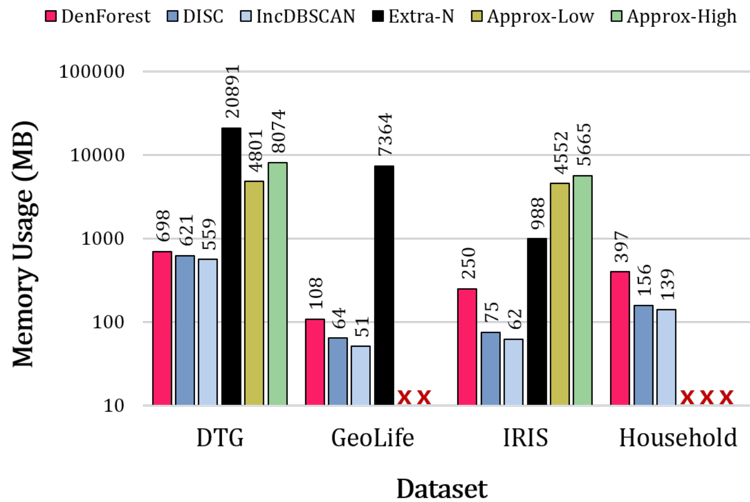


Figure 6.18: Memory usage for various datasets

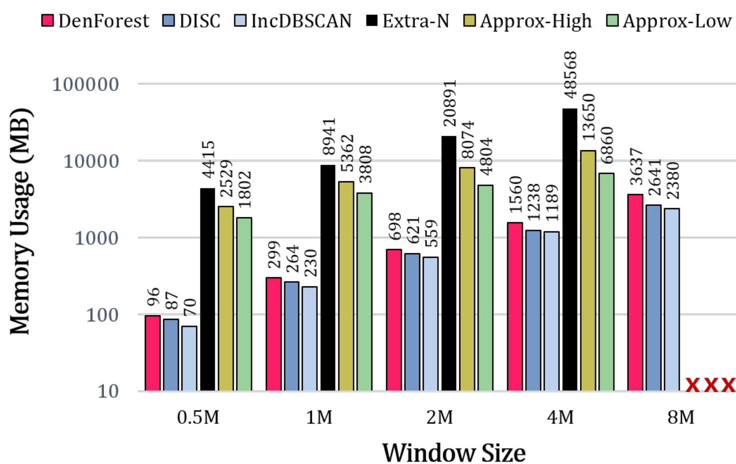


Figure 6.19: Memory usage for various window sizes (DTG)

6.5.3.3 Memory Usage

Memory usage for each method was measured for four real datasets. (See Figure 6.18.) 2M, 0.4M, 0.4M and 0.5M data points were used to fill the window for the DTG, GeoLife, IRIS, and Household datasets, respectively. The stride size was set to 5% of the window size for all datasets. Some methods did not terminate within the allotted time of ten hours denoted by a x mark in the figure.

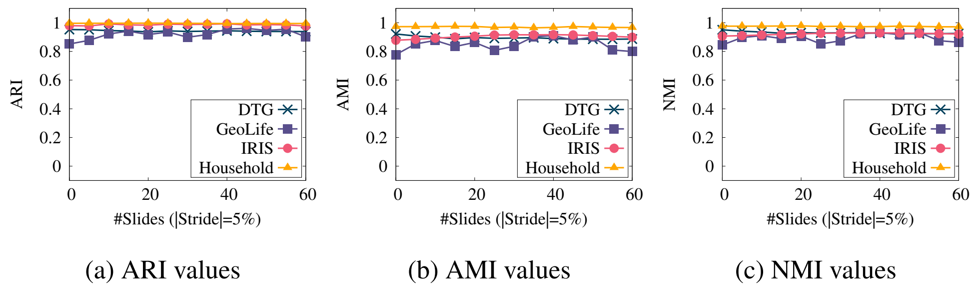
DenForest, $\mathcal{DISC}$, and IncDBSCAN have the O(N) asymptotic space complexity where N is the number of data points in the window. In practice, *DenForest* consumed more memory space than $\mathcal{DISC}$ or IncDBSCAN. This is because it uses additional memory for $\mathcal{MST}$. *DenForest* consumed less memory than Approx-Low and Approx-High. They manages grid cells to store data points and the Holm *et al.* 's data structure [44] for dynamic graph connectivity. The data structure requires at most $O(N \log N)$ space. Extra-N also consumed more memory than *DenForest*, since it manages multiple windows to avoid the deletion, which requires at most $O(N\frac{|W|}{|S|})$ space where $|W|$ and $|S|$ denote the size of the window and its stride, respectively.

Figure 6.19 shows memory usage when the window size changes from 0.5M to 8M for the DTG dataset. The trend among methods, which is shown in the previous figure, does not change as the window size increases. Approx-High/Low and Extra-N exceeds the allotted time when the window size is 8M.

6.5.3.4 Clustering Quality over Sliding Windows

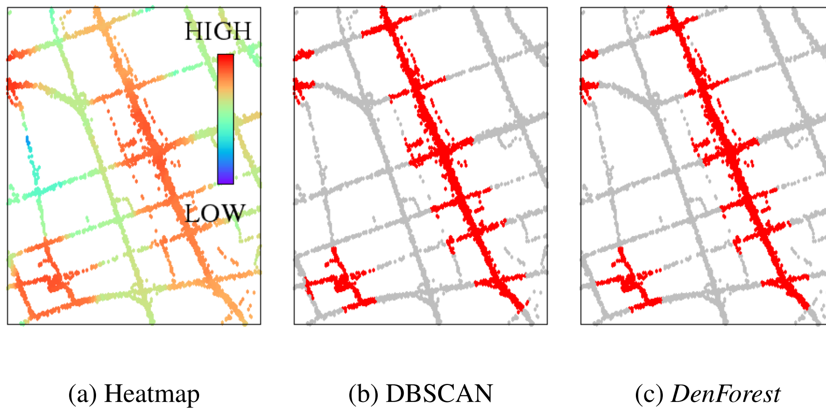
The clustering quality of *DenForest* was measured for each dataset. The true cluster labels are not available for the datasets. So the clustering results from DBSCAN were used as the ground truth. Three metrics ARI [45], AMI [83], and NMI [56] were used to measure the quality.

Figures 6.20a to 6.20c show how the clustering quality changes over time while the sliding window advances. The stride size was set to 5% of the window size. *DenForest* achieved clustering quality measurements close to one (or 100%) for all the datasets



(a) ARI values (b) AMI values (c) NMI values

Figure 6.20: Clustering quality over sliding windows

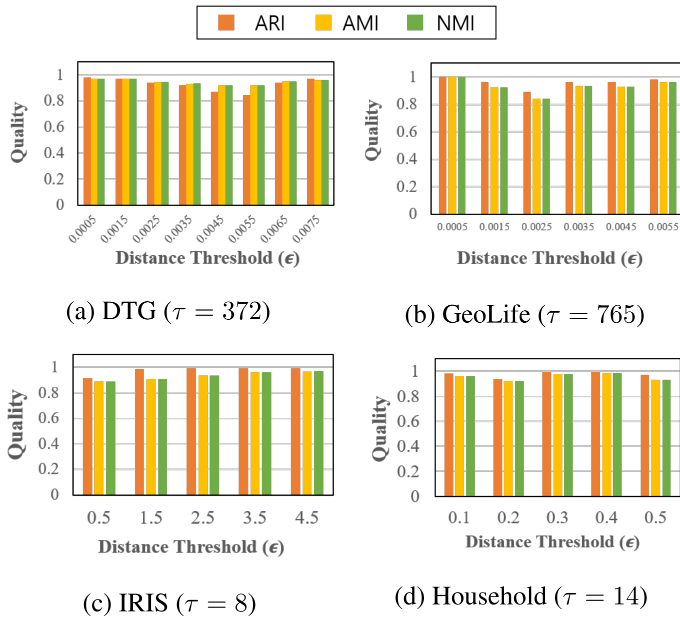


(a) Heatmap (b) DBSCAN (c) *DenForest*

Figure 6.21: Clusters found in DTG

and sustained its quality as the window slid. The average measurements of quality were 0.96 (ARI), 0.91 (AMI) and 0.93 (NMI). *DenForest* and *DenForest-NO* produce the same clustering results. Thus, their quality measurements are identical.

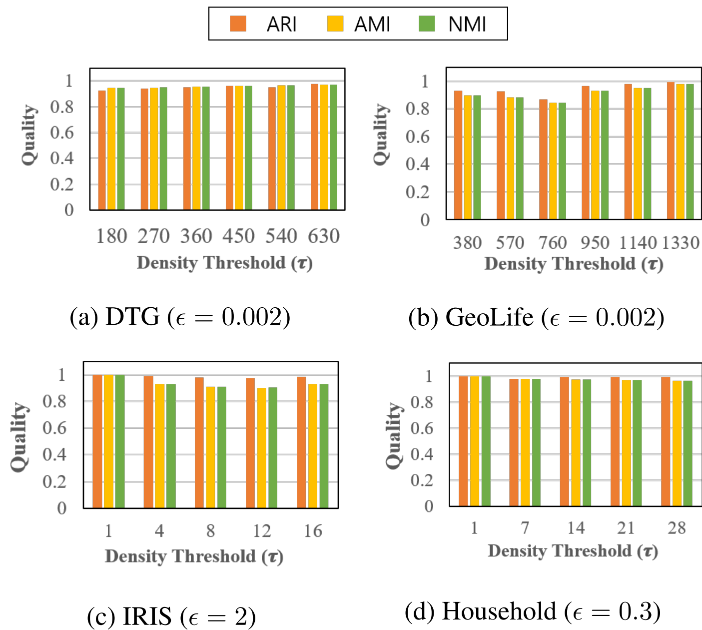
Figure 6.21 shows the heatmap and the examples of clusters detected by DBSCAN and *DenForest* for a snapshot (or window) of the DTG dataset. *DenForest* produced the result nearly identical to the result of DBSCAN, and *DenForest* detected dense areas matching well the heatmap that visualized the congested regions.



(a) DTG ($\tau = 372$) (b) GeoLife ($\tau = 765$)

(c) IRIS ($\tau = 8$) (d) Household ($\tau = 14$)

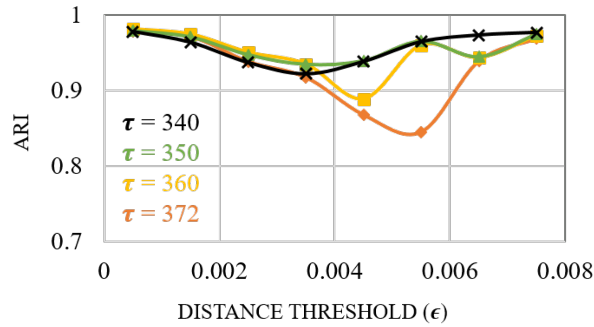
Figure 6.22: Quality under various distance thresholds



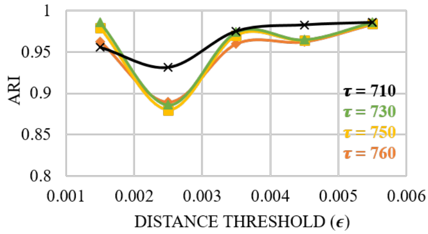
(a) DTG ($\epsilon = 0.002$) (b) GeoLife ($\epsilon = 0.002$)

(c) IRIS ($\epsilon = 2$) (d) Household ($\epsilon = 0.3$)

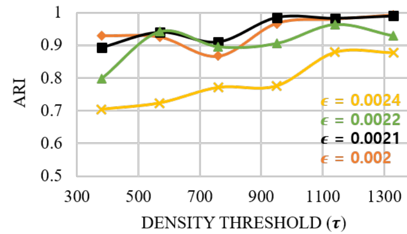
Figure 6.23: Quality under various density thresholds



(a) Relaxed τ settings for DTG



(b) Relaxed τ settings for GeoLife



(c) Relaxed ϵ settings for GeoLife

Figure 6.24: Relaxed parameter settings

6.5.3.5 Clustering Quality under Various Density and Distance Thresholds

Clustering quality of *DenForest* under various density and distance thresholds was measured. Clustering results of DBSCAN were used as the ground truth. Figure 6.22 shows the clustering quality for various distance thresholds where the density threshold for each dataset is set to the default value in Table 6.2. Figure 6.23 shows the clustering quality for various density thresholds. Similarly, the distance threshold for each dataset is set to the default value in Table 6.2.

For most settings of density and distance thresholds, *DenForest* showed high clustering quality. For some settings such as DTG (ϵ=0.0055) and GeoLife (ϵ=0.0025) in Figure 6.22 showed slightly low clustering quality. To achieve higher clustering quality, some heuristic ways are presented in the following section.

6.5.3.6 Relaxed Parameter Settings

As explained in Section 5.3.3.1, the number of nostalgic cores are related to the clustering quality. By choosing slightly lower density thresholds and higher distance thresholds than those chosen for DBSCAN, more data points become *nostalgic cores*, which contributes to improving the clustering quality.

Figure 6.24 shows the relaxed parameter settings for the DTG and GeoLife datasets. For example, in Figure 6.24a, the clustering result of DBSCAN when τ was set to the default value (372) was used as the ground truth, and it was compared with the clustering results of *DenForest* under various density thresholds. Similar experiments were also conducted using the GeoLife dataset. (See Figure 6.24b and Figure 6.24c.) As shown in Figures 6.24aand6.24b, *DenForest* achieved higher clustering quality by choosing about a 10% lower density threshold than that of DBSCAN. Moreover, *DenForest* achieved higher clustering quality by choosing about a 5% higher distance threshold than that of DBSCAN. (See Figure 6.24c.)

6.5.4 Comparison with Summarization-Based Methods

DenForest was also compared with the summarization-based methods. DBSTREAM [41] is chosen because it is shown to achieve high quality in the previous study [14]. EDMStream [37] is a streaming version of the static density peak clustering algorithm [69]. SDStream [68] and StreamSW [74] are designed for the sliding window model based on *EHCF* [93] and grids [80], respectively.

The *unlabeled real* DTG and the *labeled synthetic* Maze datasets used in Section 6.4.4 were used in the evaluation, and three metrics (ARI, AMI, and NMI) were applied to measure the quality with various window sizes. For the unlabeled DTG dataset, the clusters produced by DBSCAN were used as the ground truth. The parameters for EDMStream, DBSTREAM, SDStream, and StreamSW were tuned so as to achieve the highest quality for each window size. The update latency of one stride was also measured when the stride size was set to 5% of the window size. Only the in-

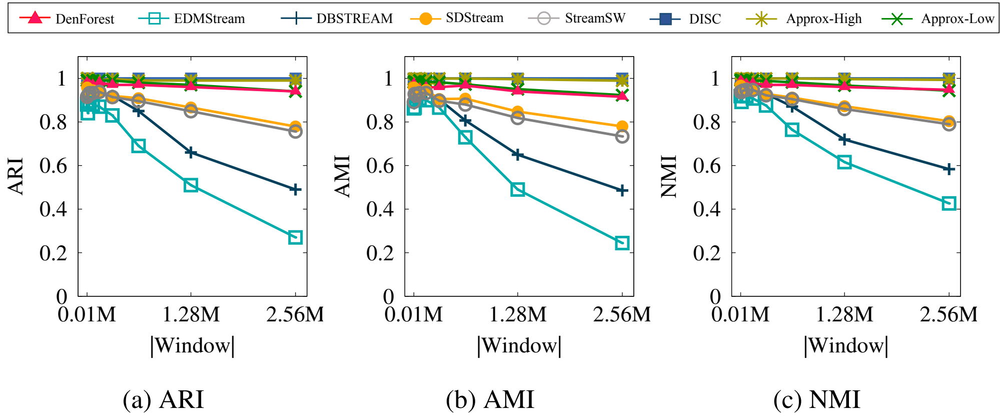


Figure 6.25: Quality of DTG clusters

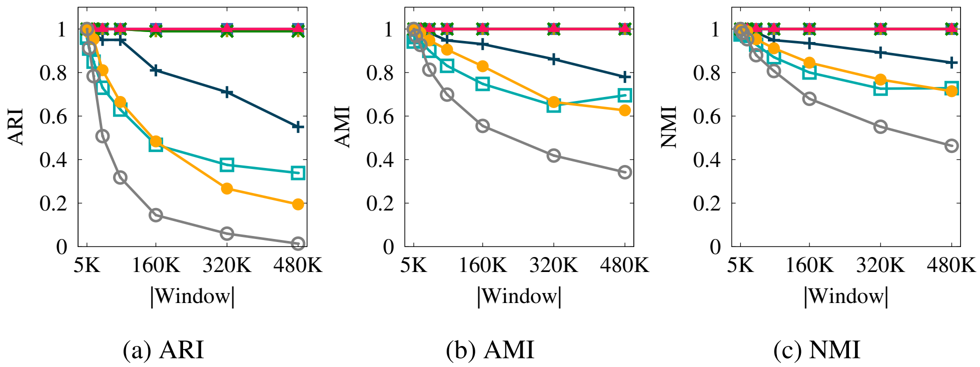
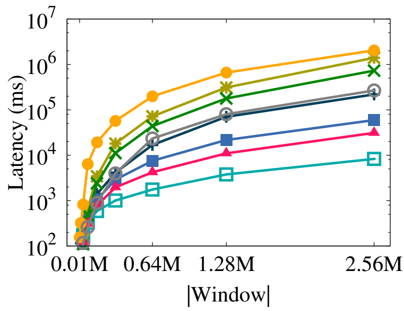


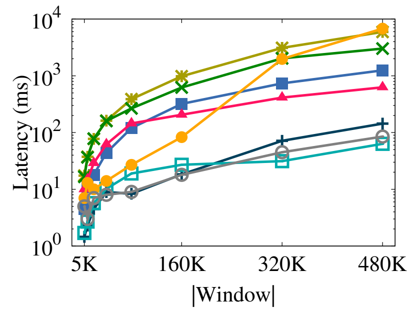
Figure 6.26: Quality of MAZE clusters

sertion latency was included in the measurements for EDMStream and DBSTREAM, because they do not support a deletion operation.

The summarization-based methods assume an infinite length of data streams and summarize a group of data points into a *micro-cluster*. Since they only maintain coarse-grained information, quality of these methods decreased steeply as the window size increased as is shown in Figures 6.25 and 6.26. Although SDStream and StreamSW achieved relatively higher quality than other summarization methods for the DTG dataset, their quality was still lower than that of *DenForest*. Moreover, they were far slower than *DenForest* due to the high cost of maintaining a number of micro-clusters



(a) DTG



(b) MAZE

Figure 6.27: Latency with DTG and MAZE

(in Figure 6.27). Conversely, *DenForest* attained high quality based on all the data points without approximation, and achieved the best performance among the methods whose quality was higher than 0.9.

Chapter 7

Future Work: Extension to Varying/Relative Densities

In this chapter, future work for density-based clustering over sliding windows as well as expected challenges are discussed.

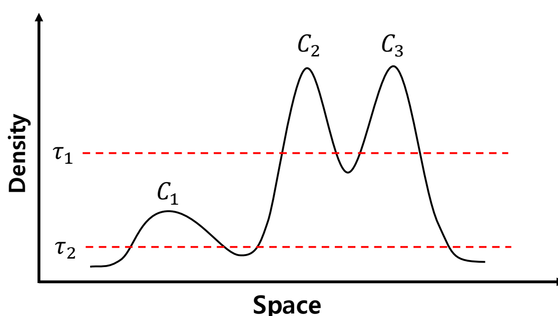


Figure 7.1: Clusters of Varying Densities

As mentioned in Chapter 2, DBSCAN [28] has a weakness in that it is hard to detect meaningful clusters of varying or relatively different densities. For example, there exist three clusters C_1, C_2, and C_3 in figure 7.1. If the density threshold of DBSACN is set to τ_2, C_2 and C_3 are detected as a single cluster. If the density threshold is set to τ_1, C_1 is not detected. To address this problem for the static datasets, OPTICS [5] and HDBSCAN [12] were proposed.

Similar to DBSCAN, $\mathcal{DISC}$ and *DenForest* suffer from the same problem. Therefore, one important future work is to efficiently detect clusters having varying densities

over sliding windows. To achieve this goal, two approaches are worthy to be considered. (See Figure 7.2 for illustration.)

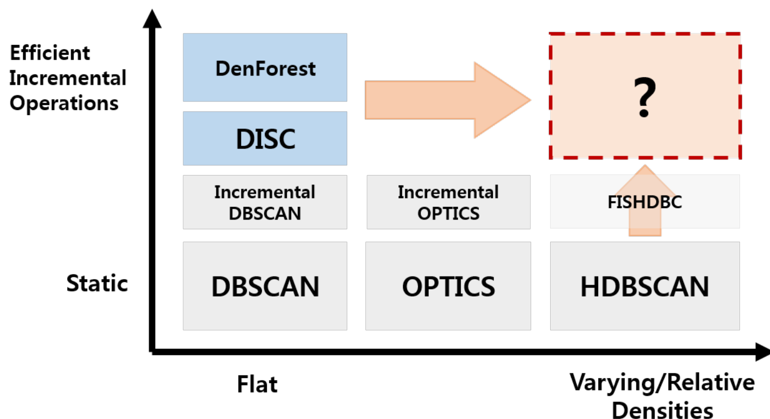


Figure 7.2: Directions for Future Work

One approach is extending $\mathcal{DISC}$ or *DenForest* to detect those clusters. A naive algorithm sketch is as follows. First, the *DenForest* or $\mathcal{DISC}$ algorithms are used to compute multiple density-based clusters of various distance or density thresholds. Those multiple density-based clusters will be the candidate clusters. Then, meaningful clusters are found by examining those candidates. Two challenges are expected in this approach. The first one is that the multiple density-based clusters should be computed without an excessive amount of computational cost. The second one is that an appropriate method is required to find the meaningful clusters.

Another approach is extending HDBSCAN or OPTICS to support efficient incremental operations. Incremental OPTICS [55] was proposed to incrementally update the result of OPTICS by supporting insert and delete operations. However, it suffers a slow deletion problem similar to Incremental DBSCAN. Moreover, it requires user's intervention to select meaningful clusters. Recently, one work [24] that incrementally detects the density-based cluster of varying densities has been proposed, but it only supports insertion. Obviously, enabling deletion will be the major challenge in this approach.

Chapter 8

Conclusion

The task of density-based clustering over sliding windows becomes an essential tool of increasing importance for data analytics given the prevalence of IoT devices. However, it is hard to be performed in real time without compromising the clustering quality.

To address the limitation of density-based clustering squarely, this dissertation proposes two algorithms $\mathcal{DISC}$ and *DenForest*. The first algorithm $\mathcal{DISC}$ efficiently detects density-based clusters over sliding windows and produces the same clustering results as DBSCAN. With COLLECT and CLUSTER operations, $\mathcal{DISC}$ can handle multiple points together. These operations are based on three key concepts, *minimal bonding cores*, *epoch-based probe*, and *multi-starter BFS*, which improve the performance by avoiding redundancy issues when updating clusters.

The second clustering algorithm mainly addresses the connectivity check problem in the deletion. *DenForest* introduces a novel concept called *nostalgic cores*, which are positioned relatively on a recent side of the window but spatially cover the $cores$ of DBSCAN. Based on the *nostalgic cores*, *DenForest* manages each cluster with *DenTree* which enables graph traversals to be avoided in the deletion. *DenForest* shows the amortized logarithmic time complexity for the deletion, which is far more efficient than previous approaches. Furthermore, the batch-optimized update operations reduce the edge updating cost in *DenTrees*.

With the extensive evaluations using various datasets, the effectiveness of those ideas was proved, and it was confirmed that $\mathcal{DISC}$ and *DenForest* can perform the clustering tasks for streaming data promptly without compromising the clustering quality. Particularly, *DenForest*'s deletion cost was almost negligible in some datasets and was less affected by the dimensionality.

This is a significant contribution given that the slow deletion problem has been a notoriously difficult challenge for density-based clustering algorithms since DBSCAN was introduced a quarter century ago. $\mathcal{DISC}$ and *DenForest* are expected to support many data analytic tasks in the streaming environment by clustering time-varying data efficiently at a low computational cost.

Bibliography

[1] AGGARWAL, C. C., HAN, J., WANG, J., AND YU, P. S. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th VLDB Conference* (Berlin, Germany, 2003), p. 81–92.

[2] AGGARWAL, C. C., WOLF, J. L., YU, P. S., PROCOPIUC, C., AND PARK, J. S. Fast Algorithms for Projected Clustering. In *Proceedings of the 1999 ACM SIGMOD Conference* (Philadelphia, Pennsylvania, USA, 1999), p. 61–72.

[3] AGRAWAL, R., GEHRKE, J., GUNOPULOS, D., AND RAGHAVAN, P. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD Conference* (Seattle, Washington, USA, 1998), p. 94–105.

[4] ALPERT, C. J., KAHNG, A. B., AND YAO, S.-Z. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics 90*, 1 (1999), 3–26.

[5] ANKERST, M., BREUNIG, M. M., KRIEGEL, H.-P., AND SANDER, J. OPTICS: Ordering Points to Identify the Clustering Structure. In *Proceedings of the 1999 ACM SIGMOD Conference* (Philadelphia, Pennsylvania, USA, 1999), p. 49–60.

[6] ARMENATZOGLOU, N., AND PAPADIAS, D. Geo-social networks. In *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, New York, NY, 2018, pp. 1620–1623.

[7] ARYA, S., AND MOUNT, D. M. Approximate range searching. *Computational Geometry 17*, 3 (2000), 135–152.

[8] BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., AND WIDOM, J. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM PODS Conference* (Madison, Wisconsin, 2002), pp. 1–16.

[9] BENTLEY, J. L. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM 18*, 9 (Sept. 1975), 509–517.

[10] BENTLEY, J. L. Decomposable searching problems. *Information Processing Letters 8*, 5 (1979), 244–251.

[11] BOHM, C., RAILING, K., KRIEGEL, H.-P., AND KROGER, P. Density connected clustering with local subspace preferences. In *Fourth IEEE ICDM Conference* (2004), pp. 27–34.

[12] CAMPELLO, R. J. G. B., MOULAVI, D., AND SANDER, J. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining* (Berlin, Heidelberg, 2013), J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, Eds., Springer Berlin Heidelberg, pp. 160–172.

[13] CAO, F., ESTER, M., QIAN, W., AND ZHOU, A. Density-Based Clustering over an Evolving Data Stream with Noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining* (Bethesda, MD, USA, 2006), pp. 328–339.

[14] CARNEIN, M., ASSENMACHER, D., AND TRAUTMANN, H. An Empirical Comparison of Stream Clustering Algorithms. In *Proceedings of the Computing Frontiers Conference* (Siena, Italy, 2017), p. 361–366.

[15] CHANG, H., AND YEUNG, D.-Y. Robust path-based spectral clustering. *Pattern Recognition 41* (01 2008), 191–203.

[16] Chen, Y., and Tu, L. Density-based Clustering for Real-time Stream Data. In *Proceedings of the 13th ACM SIGKDD Conference* (San Jose, California, USA, 2007), pp. 133–142.

[17] Chen, Y., and Tu, L. Density-Based Clustering for Real-Time Stream Data. In *Proceedings of the 13th ACM SIGKDD Conference* (San Jose, California, USA, 2007), p. 133–142.

[18] Chin, F., and Houck, D. Algorithms for Updating Minimal Spanning Trees. *Journal of Computer and System Sciences 16*, 3 (1978), 333–344.

[19] Cohen, E., and Strauss, M. J. Maintaining Time-Decaying Stream Aggregates. In *Proceedings of the 22nd ACM PODS Conference* (San Diego, California, 2003), p. 223–233.

[20] Conway, J. H., and Sloane, N. J. A. Sphere Packings and Kissing Numbers. In *Sphere Packings, Lattices and Groups*. Springer, New York, NY, 1999, pp. 1–30.

[21] Czerniawski, T., Sankaran, B., Nahangi, M., Haas, C., and Leite, F. 6D DBSCAN-based segmentation of building point clouds for planar object classification. *Automation in Construction 88* (2018), 44 – 58.

[22] Datar, M., Gionis, A., Indyk, P., and Motwani, R. Maintaining Stream Statistics over Sliding Windows (Extended Abstract). In *Proceedings of the 13th ACM-SIAM SODA Conference* (San Francisco, California, 2002), pp. 635–644.

[23] Defays, D. An efficient algorithm for a complete link method. *The Computer Journal 20*, 4 (01 1977), 364–366.

[24] Dell'Amico, M. FISHDBC: Flexible, Incremental, Scalable, Hierarchical Density-Based Clustering for Arbitrary Data and Distance. *ArXiv abs/1910.07283* (2019).

[25] How to Use a Digital Tachograph. https://www.optac.info/uk/digital-tachograph/, 2016.

[26] DUA, D., AND GRAFF, C. UCI machine learning repository, 2017.

[27] ESTER, M., KRIEGEL, H.-P., SANDER, J., WIMMER, M., AND XU, X. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proceedings of the 24th VLDB Conference* (San Francisco, CA, USA, 1998), pp. 323–333.

[28] ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd KDD Conference* (Portland, Oregon, 1996), pp. 226–231.

[29] EVERITT, B. S., AND HAND, D. J. *Mixtures of normal distributions*. Springer Netherlands, Dordrecht, 1981, pp. 25–57.

[30] FAN, T., GUO, N., AND REN, Y. Consumer clusters detection with geo-tagged social network data using DBSCAN algorithm: a case study of the Pearl River Delta in China. *GeoJournal 86* (09 2019), 317—337.

[31] FERRARA, R., VIRDIS, S. G., VENTURA, A., GHISU, T., DUCE, P., AND PELLIZZARO, G. An automated approach for wood-leaf separation from terrestrial LIDAR point clouds using the density based clustering algorithm DBSCAN. *Agricultural and Forest Meteorology 262* (2018), 434 – 444.

[32] GAMA, J. *Knowledge Discovery from Data Streams*, 1st ed. Chapman & Hall/CRC, Porto, Portugal, 2010.

[33] GAN, J., AND TAO, Y. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *Proceedings of the 2015 ACM SIGMOD Conference* (Melbourne, Victoria, Australia, 2015), pp. 519–530.

[34] GAN, J., AND TAO, Y. Dynamic Density Based Clustering. In *Proceedings of the 2017 ACM SIGMOD Conference* (Chicago, Illinois, USA, 2017), pp. 1493–1507.

[35] GAN, J., AND TAO, Y. Fast Euclidean OPTICS with Bounded Precision in Low Dimensional Space. In *Proceedings of the 2018 ACM SIGMOD Conference* (Houston, TX, USA, 2018), p. 1067–1082.

[36] GIONIS, A., MANNILA, H., AND TSAPARAS, P. Clustering Aggregation. *ACM TKDD 1*, 1 (Mar. 2007), 1–30.

[37] GONG, S., ZHANG, Y., AND YU, G. Clustering Stream Data by Exploring the Evolution of Density Mountain. *Proc. VLDB Endow. 11*, 4 (2017), 393–405.

[38] GUHA, S., RASTOGI, R., AND SHIM, K. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the 1998 ACM SIGMOD Conference* (Seattle, Washington, USA, 1998), p. 73–84.

[39] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD Conference* (Boston, Massachusetts, USA, 1984), p. 47–57.

[40] HAGEN, L., AND KAHNG, A. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11*, 9 (1992), 1074–1085.

[41] HAHSLER, M., AND BOLAÑOS, M. Clustering Data Streams Based on Shared Density between Micro-Clusters. *IEEE TKDE 28*, 6 (2016), 1449–1461.

[42] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *Unsupervised Learning*. Springer New York, New York, NY, 2009, pp. 485–585.

[43] HE, Y., TAN, H., LUO, W., MAO, H., MA, D., FENG, S., AND FAN, J. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Us-

ing MapReduce. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems* (2011), pp. 473–480.

[44] HOLM, J., DE LICHTENBERG, K., AND THORUP, M. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *Journal of the ACM 48*, 4 (2001), 723–760.

[45] HUBERT, L., AND ARABIE, P. Comparing Partitions. *Journal of Classification 1* (1985), 193–218.

[46] Incorporated Research Institutions for Seismology, 2022. `http://service.iris.edu/fdsnws/event/1/`.

[47] JAIN, A. K. Data clustering: 50 Years Beyond K-means. *Pattern Recognition Letters 31*, 8 (2010), 651 – 666.

[48] JANG, J., AND JIANG, H. DBSCAN++: Towards fast and scalable density clustering. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97, PMLR, pp. 3019–3029.

[49] JARVIS, R., AND PATRICK, E. Clustering Using a Similarity Measure Based on Shared Near Neighbors. *IEEE Transactions on Computers C-22*, 11 (1973), 1025–1034.

[50] JIANG, H. Density Level Set Estimation on Manifolds with DBSCAN. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia, 2017), JMLR.org, p. 1684–1693.

[51] JIANG, H., JANG, J., AND LACKI, J. Faster DBSCAN via subsampled similarity queries. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 22407–22419.

[52] JIANG, H., JANG, J., AND LACKI, J. Faster DBSCAN via subsampled similarity queries. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 22407–22419.

[53] KAUFMAN, L., AND ROUSSEEUW, P. J. Partitioning Around Medoids (Program PAM).

[54] KIM, J., GUO, T., FENG, K., CONG, G., KHAN, A., AND CHOUDHURY, F. M. Densely Connected User Community and Location Cluster Search in Location-Based Social Networks. In *Proceedings of the 2020 ACM SIGMOD Conference* (Portland, OR, USA, 2020), p. 2199–2209.

[55] KRIEGEL, H., KRÖGER, P., AND GOTLIBOVICH, I. Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering. In *Data Warehousing and Knowledge Discovery, 5th International Conference, DaWaK 2003, Prague, Czech Republic, September 3-5,2003, Proceedings* (2003), Y. Kambayashi, M. K. Mohania, and W. Wöß, Eds., vol. 2737, pp. 224–233.

[56] KVALSETH, T. O. Entropy and Correlation: Some Comments. *IEEE Transactions on Systems, Man, and Cybernetics 17*, 3 (1987), 517–519.

[57] LAMSAL, R. Coronavirus (COVID-19) Geo-tagged Tweets Dataset. `http://dx.doi.org/10.21227/fpsb-jz61`, 2020.

[58] LEE, C.-H. Mining Spatio-Temporal Information on Microblogging Streams Using a Density-Based Online Clustering Method. *Expert Systems with Applications 39*, 10 (Aug. 2012), 9623–9641.

[59] LUCHI, D., LOUREIROS RODRIGUES, A., AND MIGUEL VAREJÃO, F. Sampling approaches for applying DBSCAN to large datasets. *Pattern Recognition Letters 117* (2019), 90–96.

[60] LULLI, A., DELL'AMICO, M., MICHIARDI, P., AND RICCI, L. NG-DBSCAN: Scalable Density-Based Clustering for Arbitrary Data. *Proc. VLDB Endow. 10*, 3 (Nov. 2016), 157–168.

[61] MACQUEEN, J. Some Methods for Classification and Analysis of Multivariate Observations. In *the 5th Berkeley Symposium on Mathematical Statistics and Probability Data Mining* (Berkeley, CA, USA, 1965), pp. 281–297.

[62] MARIESCU-ISTODOR, P. F. R., AND ZHONG, C. XNN graph. 207–217.

[63] METROPOLIS, N., AND ULAM, S. The Monte Carlo Method. *Journal of the American Statistical Association 44*, 247 (1949), 335–341.

[64] MOON, T. The expectation-maximization algorithm. *IEEE Signal Processing Magazine 13*, 6 (1996), 47–60.

[65] NG, A., JORDAN, M., AND WEISS, Y. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems* (2001), T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14, MIT Press.

[66] NG, R., AND HAN, J. Clarans: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering 14*, 5 (2002), 1003–1016.

[67] NTOUTSI, I., ZIMEK, A., PALPANAS, T., KRÖGER, P., AND KRIEGEL, H.-P. Density-based Projected Clustering over High Dimensional Data Streams. In *Proceedings of the 2012 SIAM International Conference on Data Mining* (München, Germany, 2012), pp. 987–998.

[68] REN, J., AND MA, R. Density-Based Data Streams Clustering over Sliding Windows. In *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery* (Tianjin, China, 2009), vol. 5, pp. 248–252.

[69] RODRIGUEZ, A., AND LAIO, A. Clustering by fast search and find of density peaks. *Science 344*, 6191 (2014), 1492–1496.

[70] SANDER, J., ESTER, M., KRIEGEL, H.-P., AND XU, X. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery 2*, 2 (Jun 1998), 169–194.

[71] SCHUBERT, E., SANDER, J., ESTER, M., KRIEGEL, H. P., AND XU, X. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM TODS 42*, 3 (July 2017), 1–21.

[72] SHEN, J., HAO, X., LIANG, Z., LIU, Y., WANG, W., AND SHAO, L. Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm. *IEEE Transactions on Image Processing 25*, 12 (2016), 5933–5942.

[73] SHILOACH, Y., AND EVEN, S. An On-Line Edge-Deletion Problem. *J. ACM 28*, 1 (Jan. 1981), 1–4.

[74] SHYAM SUNDER REDDY, K., AND SHOBA BINDU, C. StreamSW: A density-based approach for clustering data streams over sliding windows. *Measurement 144* (2019), 14–19.

[75] SIBSON, R. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal 16*, 1 (01 1973), 30–34.

[76] SLEATOR, D. D., AND TARJAN, R. E. A Data Structure for Dynamic Trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing* (Milwaukee, Wisconsin, USA, 1981), Association for Computing Machinery, p. 114–122.

[77] SONG, H., AND LEE, J.-G. RP-DBSCAN: A Superfast Parallel DBSCAN Algorithm Based on Random Partitioning. In *Proceedings of the 2018 ACM SIGMOD Conference* (Houston, TX, USA, 2018), pp. 1173–1187.

[78] TANGWONGSAN, K., HIRZEL, M., SCHNEIDER, S., AND WU, K.-L. General Incremental Sliding-window Aggregation. *Proc. VLDB Endow. 8*, 7 (2015), 702–713.

[79] THEN, M., KAUFMANN, M., CHIRIGATI, F. S., HOANG-VU, T.-A., PHAM, K., KEMPER, A., NEUMANN, T., AND VO, H. T. The More the Merrier: Efficient Multi-Source Graph Traversal. *Proc. VLDB Endow. 8*, 4 (2014), 449–460.

[80] TU, L., AND CHEN, Y. Stream Data Clustering Based on Grid Density and Attraction. *ACM TKDD 3*, 3 (July 2009).

[81] VEENMAN, C., REINDERS, M., AND BACKER, E. A Maximum variance Cluster Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24* (10 2002), 1273– 1280.

[82] VENKATASUBRAMANIAN, S. Clustering on Streams. In *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, 2009, pp. 378–383.

[83] VINH, N. X., EPPS, J., AND BAILEY, J. Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary? In *Proceedings of the 26th ICML Conference* (Montreal, Quebec, Canada, 2009), p. 1073–1080.

[84] WANG, Y., GU, Y., AND SHUN, J. Theoretically-Efficient and Practical Parallel DBSCAN. In *Proceedings of the 2020 ACM SIGMOD Conference* (Portland, OR, USA, 2020), p. 2555–2571.

[85] WANG, Y., YU, S., GU, Y., AND SHUN, J. Fast Parallel Algorithms for Euclidean Minimum Spanning Tree and Hierarchical Spatial Clustering. In *Proceedings of the 2021 ACM SIGMOD Conference* (Virtual Event China, 2021), p. 1982–1995.

[86] XU, X., JÄGER, J., AND KRIEGEL, H.-P. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Min. Knowl. Discov. 3*, 3 (sep 1999), 263–290.

[87] YANG, D., RUNDENSTEINER, E. A., AND WARD, M. O. Neighbor-based Pattern Detection for Windows over Streaming Data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (Saint Petersburg, Russia, 2009), pp. 529–540.

[88] YANG, M.-S., LAI, C.-Y., AND LIN, C.-Y. A robust em clustering algorithm for gaussian mixture models. *Pattern Recognition 45*, 11 (2012), 3950–3961.

[89] YANG, DI AND RUNDENSTEINER, ELKE A. AND WARD, MATTHEW O. Summarization and Matching of Density-based Clusters in Streaming Environments. *Proc. VLDB Endow. 5*, 2 (2011), 121–132.

[90] ZAHN, C. T. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers C-20*, 1 (1971), 68–86.

[91] ZHANG, T., RAMAKRISHNAN, B., AND LIVNY, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD Conference* (Montreal, Canada, 1996), pp. 103–114.

[92] ZHENG, Y., FU, H., XIE, X., MA, W.-Y., AND LI, Q. Geolife GPS trajectory dataset - User Guide, July 2011. `https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/`.

[93] ZHOU, A., CAO, F., QIAN, W., AND JIN, C. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems 15* (05 2008), 181–214.

[94] ZHU, Y., AND SHASHA, D. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proceedings of the 28th VLDB Conference* (Hong Kong, China, 2002), pp. 358–369.

초 록

모바일 및 IoT 장치가 널리 보급됨에 따라 스트리밍 데이터상에서 지속적으로 클러스터링 작업을 수행하는 것은 데이터 분석에서 점점 더 중요해지는 필수 도구가 되었습니다. 많은 클러스터링 방법 중에서 밀도 기반 클러스터링은 노이즈가 존재할 때 임의의 모양의 클러스터를 감지할 수 있다는 고유한 장점을 가지고 있으며 이에 따라 많은 관심을 받았습니다. 그러나 밀도 기반 클러스터링은 변화하는 입력 데이터 셋에 따라 지속적으로 클러스터를 업데이트해야 하는 경우 비교적 높은 계산 비용이 필요합니다. 특히, 클러스터에서의 데이터 점들의 삭제는 심각한 성능 저하를 초래합니다.

본 박사 학위 논문에서는 슬라이딩 윈도우상의 밀도 기반 클러스터링의 성능 한계를 다루며 궁극적으로 두 가지 알고리즘을 제안합니다. 첫 번째 알고리즘인 $\mathcal{DISC}$ 는 슬라이딩 윈도우상에서 DBSCAN과 동일한 클러스터링 결과를 찾는 점진적 밀도 기반 클러스터링 알고리즘입니다. 해당 알고리즘은 클러스터 업데이트 시에 발생하는 중복 문제들에 초점을 둡니다. 밀도 기반 클러스터링에서는 여러 데이터 점들을 개별적으로 삽입 혹은 삭제할 때 주변 점들을 불필요하게 중복적으로 탐색하고 회수합니다. $\mathcal{DISC}$ 는 배치 업데이트로 이 문제를 해결하여 성능을 향상시키며 여러 최적화 방법들을 제안합니다. 두 번째 알고리즘인 *DenForest* 는 삭제 과정에 초점을 둔 점진적 밀도 기반 클러스터링 알고리즘입니다. 클러스터를 그래프로 관리하는 이전 방법들과 달리 *DenForest* 는 클러스터를 신장 트리의 그룹으로 관리함으로써 효율적인 삭제 성능에 기여합니다. 나아가 배치 최적화 기법을 통해 삽입 성능 향상에도 기여합니다. 두 알고리즘의 효율성을 입증하기 위해 광범위한 평가를 수행하였으며 $\mathcal{DISC}$ 및 *DenForest* 는 최신의 밀도 기반 클러스터링 알고리즘들보다 뛰어난 성능을 보여주었습니다.

주요어: 밀도 기반 클러스터링, 데이터 스트림, 슬라이딩 윈도우, 삭제, 점진적 클러스터링, DISC, DenForest
학번: 2015-22894